GENERATION OF LINUX COMMANDS USING NATURAL LANGUAGE DESCRIPTIONS

A Thesis
Presented to
The Academic Faculty

By

Ananya Raval

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science in Computer Science

Georgia Institute of Technology

May 2018

GENERATION OF LINUX COMMANDS USING NATURAL LANGUAGE DESCRIPTIONS

Approved by:

Professor Devi Parikh, Advisor
School of Interactive Computing
Georgia Institute of Technology

Professor Dhruv Batra
School of Interactive Computing
Georgia Institute of Technology

Professor Duen Horng Chau
School of Computational Science and Engineering
Georgia Institute of Technology

Date Approved: January 11, 2018

All things are difficult before they are easy.

Dr. Thomas Fuller

I dedicate this work to my parents, brother and teachers.

ACKNOWLEDGEMENTS

I want to express my heartfelt gratitude to Prof. Devi Parikh for taking on my project and seeing it through the end. I want to thank Dr. Dhruv Batra and Dr. Duen Horng Chau for reviewing this thesis and providing insightful comments.

I sincerely thank Ramakrishna Vedantam for his perennial guidance on all aspects of my project. Exploring the data, laying out a structure for experiments to be conducted, training neural networks - his inputs were invaluable. His comments helped shape this document immensely.

Amitesh Sinha, Arjun S. Pejathya, Mohit Kuri, Pratik, Prathyakshun Rajashankar, Buggana Sathvik Sanjiv, Taruneshwar Kandula created this dataset. They put in a lot of time and effort writing the commands, checking their validity and writing their descriptions. I thank them for all the hard work.

My parents are my strongest pillars of support. They are a constant source of my inspiration. I express gratitude to my elder brother for supporting and guiding me throughout graduate school.

My lifelong friends Avani Pallath, Aditya Khetan and Prakash Murali are the best people I have ever met. They have been by my side through thick, thin and everything in between. Graduate school would be a dull and boring place without Aditi Gupta, Vasavi Gajarla, Shashidhar Ravishankar and Sai Teja. They made Georgia Tech fun.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Translating natural language into source code or programs is an important problem in natural language understanding – both in terms of practical applications and in terms of understanding usage of language to affect action. In this domain, we consider the problem of translating natural language descriptions of LINUX commands into the corresponding commands. This is useful from the point of view of users who want to get commands executed but lack expertise to come up with them on the bash terminal. The major contribution of this thesis is a parallel corpus for translating natural language into LINUX commands. The corpus contains 4561 unique commands and 3-4 descriptions for each command, making a total of 11177 pairs. Along with the corpus, simple classification settings using Support Vector Machines and translation settings using Sequence to Sequence Recurrent Neural Network based models are studied to provide benchmarks for machine learning model performance on the collected dataset. This document provides analysis of the collected dataset, and describes the results and findings from models trained on the dataset.

CHAPTER 1
INTRODUCTION AND BACKGROUND

1.1 Introduction

Automatic generation of code is a long studied problem in Artificial Intelligence. Early work by Waldinger et al.[1] takes input specifications in the form of predicate calculus and produces a LISP program, Manna et al.[2] approaches program synthesis as a theorem proving task and outlines a deductive approach combining unification, mathematical induction and transformation rules. Creating such input specifications is tedious, sometimes more than generation of the program itself.

Recently, machine learning and deep learning are being used for a variety of tasks like learning complex behavior of programs and differentiable data structures [3], probabilistic programming and program synthesis using input/output examples [4, 5, 6]. Such efforts have many practical applications: building tools assisting in software development by predicting tokens and API calls [7, 8, 9, 10, 11], summarizing source code [12, 13] etc.

1.2 Motivation

Automatic construction of programs based on natural language specifications is an intriguing and challenging problem. Many programming tasks are repetitive and require the developer to learn a domain-specific language to deliver them. Motivated by the need to ease this task, we propose to generate Linux commands with the intent expressed in natural language.

Linux commands are ubiquitously used. They have restricted structure and functionality as they are inherently function calls to the bash shell. A user new to Linux might not be familiar with the commands, but can express the desired functionality in natural language

Figure 1.1: Motivation

instead of spending time going through the Linux manual in order to execute the correct command. Such a framework would be useful for all kinds of end-users, including developers and general public who are Linux users without having to learn shell commands. Figure 1.1 is a visual representation of this goal. To the best of our knowledge, [14] is the only other work towards achieving this goal. They perform translation using recurrent neural networks enhanced with slot filling.

1.3 Our Contribution

- *Data*: We choose 47 commonly used Linux Commands. We create executable samples for each command using it's supported options and arguments. We create 3-4 natural language specifications for each sample describing it's function in detail.

- *Model*: We pose the generation of commands as a translation problem from natural language to executable Linux examples. We model the translation problem using a sequence to sequence recurrent neural network model ([15]) and report how well we are able to translate natural language to Linux commands.

1.4 Related Work

There have been many efforts to model source code with natural language as input.

[13] uses 500 examples of 4 programs and demonstrates that with enough data, syntax and intent of the program can be understood. Automated translation of input program specifications to C++ code is explored by modeling the problem as a joint dependency parsing and semantic role labeling task by [12]. Our approach is similar to these in terms of the amount of supervision. However, we use sequence to sequence networks[15] instead of a simple RNN and the output generated is the Linux command, instead of a parser.

A Neural Programmer [16] is a neural network augmented with a set of discrete operations and creates a program made up these operations along with the result of running this program against a database table. Latent Predictor Network[17] is a novel neural architecture which generates an output sequence by marginalizing multiple predictors. These predictors are a mix of natural language and structured specifications. Their performance is shown on collectible trading card games. [18] presents a novel neural network powered by a grammar model to explicitly capture a target syntax. They define a probabilistic grammar model to generate an AST given natural descriptions and deterministically convert it into code. Work done by Kevin Guu et. al[19] attempts to learn semantic parser to map natural language utterances to executable programs using only indirect supervision. In contrast to these approaches, we generate commands using only natural language descriptions and no other signals.

Work done by [3] provides a framework for constructing program synthesizers that take natural language (NL) inputs and produce expressions in a target DSL is another work. In contrast, our work is specific to Linux commands. Attempts by [12, 13] are in the reverse direction with input as code and output being a natural language summary of source code.

Our work is an attempt towards generating executable Linux commands using it's complete and comprehensive description. We present a novel dataset containing pairs of executable Linux commands and their corresponding natural language descriptions. Linux commands we chosen as they are ubiquitously used among all users and they have restricted structure and function. Their descriptions can be constructed very accurately using freely available Linux manuals. We pose this problem as machine translation problem as each pair can be thought of as two sentences depicting the same intent/meaning. The natural language description describes the complete function of a command and the command contains complete functionality for each intent mentioned in the description.

CHAPTER 2

THE NATURAL LANGUAGE TO LINUX COMMAND DATASET

Our primary objective is to translate natural language descriptions (NL) into Linux commands. For this purpose, we chose 47 Linux commands which are commonly used by Linux users. For each Linux command, we chose a small set of options provided in the Linux Manual, created a comprehensive list of executable commands using these options and collected 3-4 natural language descriptions for each command.

2.1 Data Collection

For the purpose of this problem, we create data in the following manner: sample complete commands for each of the 47 Linux commands, collect 3-4 natural language descriptions which describe the complete functionality of each sample command and execute each command to ensure that the input-output relationship of interest is satisfied. Data collection is easier this way as opposed to first collecting user intent for a task in the form of natural language description and understanding which command should be used to execute the intent and verifying that the intent is indeed satisfied.

The data collection was done by undergraduate users from different institutions. Because of the specialized nature of this task, each user was assigned a set of Linux commands, so they could become expert in the syntax and flags of a particular set, making the process more efficient. Table A.1 in the Appendix shows the sets of Linux commands assigned to each user.

2.2 Datasets and Tasks

We run experiments for both classification and translation on the data collected. For each task, we create two datasets - a small dataset and a complete dataset.

2.2.1 Datasets for Classification

As mentioned in section 2.1, each sample command in the corpus is an executable unit of code for one built-in Linux command. We can consider this Linux command as a *Base Class*, as it defines the main goal of the complete sample command and of it's corresponding description. Table 2.1 shows a few examples of sample commands, their descriptions and their true base class.

Hence, we can categorize each natural language description to one of the 47 base classes. Primary motivation of this exercise is to determine whether a description can identify the main purpose of the user by determining the correct Linux command to be used.

Table 2.1: Base classes of commands and descriptions

Command	**Description**	**Base Class**
tail -n7 lol.txt	Show last 7 lines of lol.txt.	tail
rm *	Empty this folder by deleting all files.	rm
cp new1.txt /	Copies new1.txt to root directory.	cp

1. *Small Dataset* (CL-21): This dataset consists of the commands/classes shown in Table 2.2.

Table 2.2: CL_21

awk	bzip2	cat	cd	chown	cp	echo
export	getfacl	history	kill	less	ln	ls
mkdir	mv	pwd	rm	sed	sleep	uniq

2. *Complete Dataset* (CL-42): This dataset consists of all classes/commands except mount, service, sort, tar, free. We did not include these commands because majority of the

descriptions contain the name of the class, which in turn give away the class information. It is challenging to describe complete commands for these classes without using these terms themselves. Table 2.3 shows example pairs for these Linux commands.

Table 2.3: Example commands not included in CL-42

Command	Descriptions
mount -t test1, test2	– Mount file systems of the types test1 and test2. – How to mount the mount points of type test1 and test2? – Single command to mount the file systems of types test1 and test2.
service –full-restart	– Restart all the running services in the system. – How can I restart all the services that are currently running? – Command to restart the services running at the moment on the system. – How can I make all the services in the system to restart?
sort -nk2,2 nums.txt	– Do numeric sort on the lines of file nums.txt base on second key. – Perform numeric sort on nums.txt and use second word as key. – Order the lines of nums.txt treating them as numbers and use 2nd word for sorting.
tar –no-xattrs -cjf last.tar.bz2 text.txt	– Create bxip2 archive, without extended attribute support, last.tar with text.txt in it. – Archive the text.txt file inside last.tar.bz2 bzip2 archive without extended attribute support. – Put text.txt in new bzip2 archive last.tar.bz2 without extended attribute support.
free –si	– How much free memory do I have on my disk? Use powers of 1000 not 1024.

Table 2.3 Continued from previous page

Command	Descriptions
	– Show the amount of free memory on my computer. Divide by 1000 not 1024. – Display free RAM memory. Divide by 1000 not 1024.

2.2.2 Datasets for Translation

We created 2 types of datasets, based on the amount of overlap between the commands in the train test and development set.

COMPOSITIONAL DATASET: In compositional datasets, there is no overlap between the test, train and validation sets. This is a form of a test for compositional generalization for the models: can the model learn the syntax and options for a particular Linux command and generate an executable command even if the exact same command has not been seen in the training set?

1. *Small Dataset* (COMPOSITIONAL-40): We create this dataset by randomly choosing 40% of all the commands. Out of this 40%, we created train, validation and test data with the ratio 70:10:20.

2. *Complete Dataset* (COMPOSITIONAL-100): This dataset is a union of all commands from all LINUX Commands. We create train, validation and test data with ratio 70:10:20.

IID DATASET: As mentioned before, each command contains about 3-4 descriptions on an average. In IID datasets, we take all pairs of (Command,Description) and divide them instead of dividing individual commands. Hence, commands can be common between train,test and validation data.

1. *Small Dataset* (IID-40): We create this dataset by randomly choosing 40% of all pairs. Out of this 40%, train, validation and test are in ratio 70:10:20.

2. *Complete Dataset* (IID-100): We take all pairs of commands and descriptions from all LINUX Commands. The same ratio, 70:10:20 is used for creating train,test and validation data.

2.3 Summary Statistics

This section includes a snapshot of data and statistics such as Linux command distribution and length of descriptions.

2.3.1 Snapshot of Data

Table 2.3 contains a examples of a few commands and their descriptions. We chose these commands to reflect the variety of (Command, Description) pairs in terms of length of commands, length and number of descriptions, different number of combinations of options and different number of arguments for a complete command.

Table 2.4: Snapshot of Data

Command	Descriptions
awk '$1 ~ /hello/' marks.txt	– Print lines of marks.txt containing 'hello' word in column 1. – How do I print records containing word 'hello' in column 1 from marks.txt? – How to display lines from marks.txt containing 'hello' in column 1? – Show records from marks.txt containing 'hello' word in column 1.
bzip2 -z -1 file.txt	– How do I force compress file.txt using level 1 compression? – Compress the file file.txt with level 1 compression forcefully. – Compress file file.txt forcefully in this folder. Use level 1 compression.

Table 2.4 Continued from previous page

Command	Descriptions
cat -be instructions.txt	– Display file instructions.txt with line numbers for non empty lines, non-printing characters and $ symbol at end of each line. – How do I see the content of file instructions.txt with line numbers for non-blank lines, non-printing characters and $ symbol at end of each line? – Open file instructions.txt. Show line numbers only for non empty lines. Show non-printing characters and $ at end of each line. – Show the content of instructions.txt with numbering of each non-blank line, non-printing characters and $ symbol at end of each line. – In the command line how do I see contents of file instructions.txt with line numbers for only non-blank lines, non-printing characters and $ at end of each line.
cd -L /relative/path/to/folder	– Change current environment to /relative/path/to/folder and force symbolic links to follow. – Make my current workspace to /relative/path/to/folder. Force symbolic links to be followed. – Command to change directory to /relative/path/to/folder. Force symbolic links to be followed. – Move to folder /relative/path/to/folder and force symbolic links to follow.
chown -v myuser:mygroup myfile	– Change the user ownership to myuser and group ownership to mygroup for the file myfile. Display the ownership details of myfile. – Show the ownership of file myfile after making the change in user ownership/group ownership to myuser/mygroup.

Table 2.4 Continued from previous page

Command	Descriptions
	– How do I know who are the files' owners when I try changing user ownership to myuser and group ownership to mygroup in myfile? – Show verbose explanation when changing the user ownership to myuser and group ownership to mygroup in myfile.
chmod -Rf u=rwx,g=rx,o=r foldername/	– Change permissions for all the content directory in foldername. User can read, write and execute. Group can read and execute it. Others can only read it. Prevent error messages from being shown. – Change permissions for all the content folder in foldername. User can read, write and execute. Group can read and execute it. Others can only read it. Prevent error messages from being shown. – How do I change permissions of all content in directory foldername so that user can do everything, group can only read and execute and others can only read it and Prevent error messages from being shown. – How do I change permissions of all content in folder foldername so that user can do everything, group can only read and execute and others can only read it and Prevent error messages from being shown.
cp –backup=simple *.txt ../office	– Create a backup of each existing destination file in office folder in parent directory while copying all files ending with .txt. – How to copy all files ending with .txt to ../office and create a backup for existing destination files in relative path ../office?

Table 2.4 Continued from previous page

Command	Descriptions
	– How to keep the existing files instead of overwriting while copying all .txt files to office folder in parent directory?
df -BE	– Display statistics of file system in Exabytes. – Display device name,total blocks,total disk space,available disk space and mount points of the system in Exabytes. – How do I display statistics of file system in terms of Exabytes? – How do I display device name, total blocks, total disk space, available disk space and mount points of the system in Exabytes?
diff -pl aa.cpp AA.cpp	– Show the differences in C source files aa.cpp and AA.cpp and paginate the output. – Paginate the difference between aa.cpp and AA.cpp C files. – Output the distinctions between aa.cpp and AA.cpp C files and ready the output for printout.
du –inodes -t10K /home/user/	– Displays the memory occupied by the folders and sub-folders in /home/user/ in terms of inodes by ignoring the folders having size less than 10K. – How do I view the memory occupied by the folder and the sub-folder in /home/user/ in terms of inodes and ignore the folders having size less than 10K? – View the memory occupied by the folder and the sub-folder in /home/user/ in terms of inodes. Ignore the folders which have size less than 10K. – How do I show the memory occupied by the folders and sub-folders in /home/user/ in terms of inodes by ignoring the folders which have size less than 10K?

Table 2.4 Continued from previous page

Command	Descriptions
echo *	– Print all file names in current folder. – What files are present in this folder? – How do I find out the files present in this folder? – List all files in this folder.
sort -bMr random.txt	– Reverse sort the lines in the file random.txt ignoring starting white spaces and sort the months in year. – Order the contents of random.txt ignoring starting blanks and in reverse order of months of year. – Arrange the contents of random.txt in reverse order of months and ignoring leading spaces.

2.3.2 Command Distribution

The number of complete commands for each Linux command varies highly depending on a number of factors. Technical factors include the amount of options available, the number of valid combinations of options and the number of commands which can be executed properly on a personal Linux machine. Practical factors include the amount of time and number of people available.

Figure 2.1 shows the Command distributions of all 47 commands split into two plots for clear visibility. We can see that the distribution of the number of complete commands is somewhat skewed based on the factors mentioned above. The minimum number of commands is 4 for history and the maximum number of commands is 1069 for mount.

Figure 2.2 shows the box plot of the command distribution.The box plot for command shows that about 50 percentile of data lies in the range 10-100, with 2 outliers.

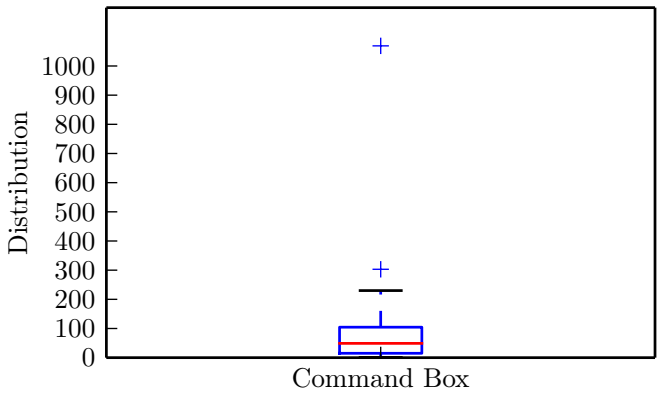
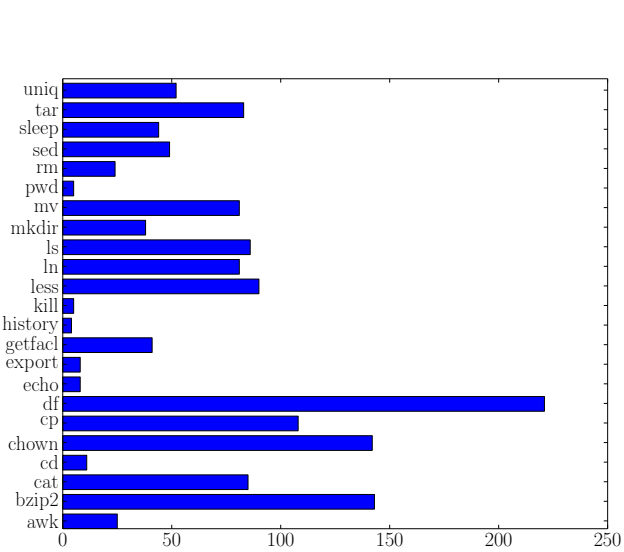
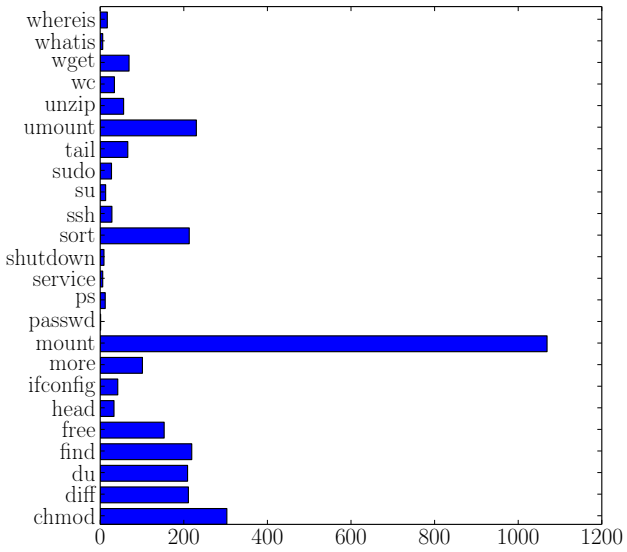


Figure 2.1: Number of Commands



(a) Distribution of 22 commands



(b) Distribution of 25 Commands

Figure 2.2: Command distribution

2.3.3 Length of Descriptions

The length of natural language descriptions is important for translation. The longer the description, the more challenging it is to identify and keep the relevant contents from the description in memory.

Figure 2.3 depicts the length distribution of all descriptions. Most of the descriptions have length in range 10-30, as shown in the histogram. There are many descriptions with

length is greater than 45 words. These descriptions are those which contain $>= 4$ options in the corresponding commands.

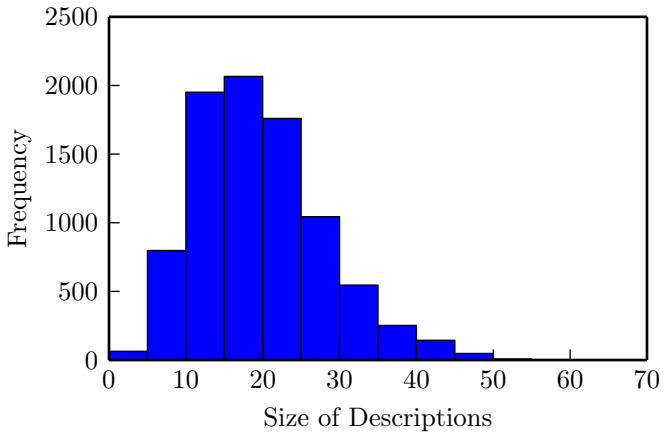


Figure 2.3: Distribution of Length of Descriptions

2.3.4 Similarity of Descriptions

For each Linux command, we explore the correlations between all descriptions including a few questions. Measuring lexical similarity provides an insight about the variety of words used while creating these descriptions. This in turn would help us appreciate how much invariance we need to learn when mapping descriptions to commands.

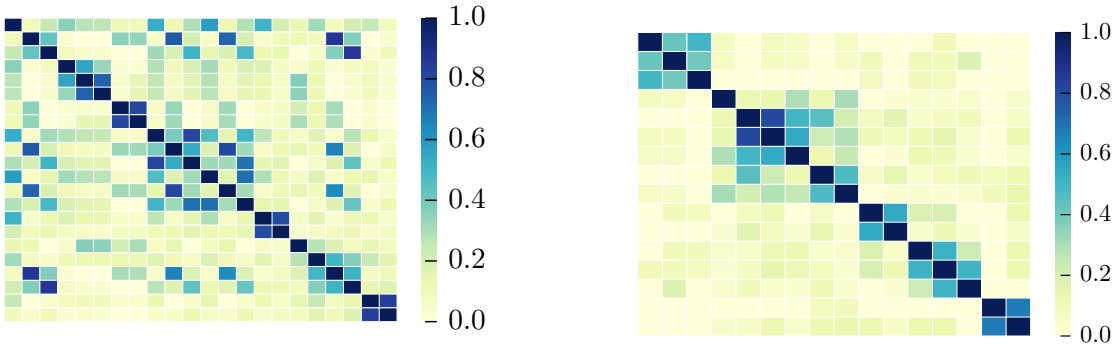
We create heatmaps of cosine similarities for each Linux command where each description vector is a Bag of Words representation using either sklearn or nltk dictionaries.

Figure 2.4 shows heatmaps of ps and export commands. There is a high density of similarities $<$0.5.

Figure 2.5 shows heatmaps of su and echo commands. sleep command contains a lot of descriptions with an almost equal mix of low and high correlation values. diff command is similar to sleep command with a high density of correlation values $>$0.5.

Figure 2.6 shows heatmaps of wc and whereis commands. Both commands contain descriptions with a very high density of correlation values $>$0.8.

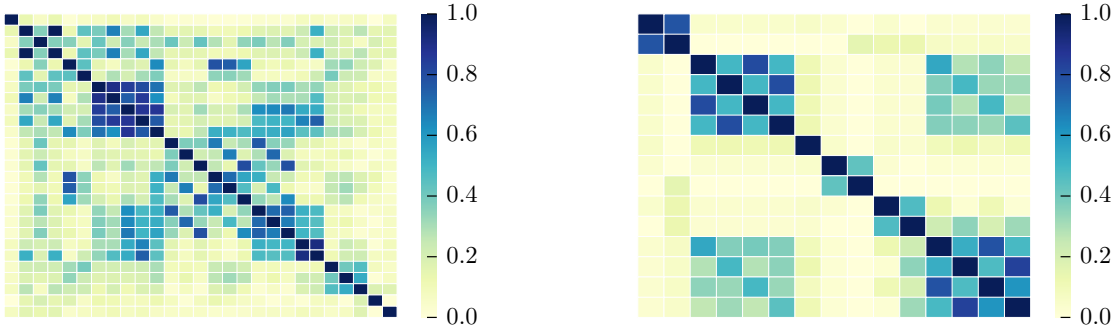
Table 2.5 shows a few descriptions with different correlation values.



(a) ps

(b) export

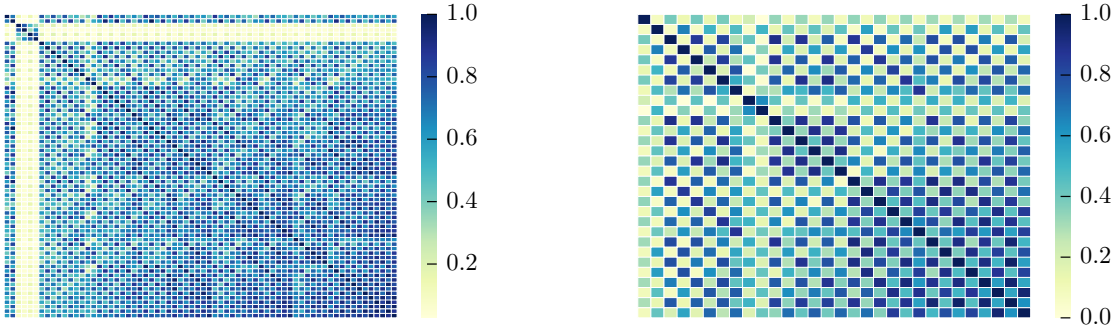
Figure 2.4: Low correlation values



(a) su

(b) echo

Figure 2.5: Medium correlation values



(a) wc

(b) whereis

Figure 2.6: High Correlation values

The x- and y- axes are ordered descriptions of the corresponding command in the figures.

Table 2.5: Descriptions and Correlations

Description 1	**Description 2**	**Correlation**
Display column 4 then followed by tab space then by column 3 of marks.txt. Display only those which contain ”a” in records.	Print contents of table.csv with line number for each record.	0.0154318428728
Make folder with name mydir. Set permissions:read,write and execute for all users. Set security context as default type.	Create the ananya folder and it’s parent folders(myfolder/main) if absent. Set its permissions such that owner can read, write and execute the contents but all others can only execute.	0.200366128534
List all contents including hidden ones present in the Documents directory in a comma separated list.	Show detailed list of files and folders of the directory titled Videos, including hidden ones. Do not sort or colorize the files.	0.300143842301
Substitue foo with bar in file.txt if the line contains ’baz’.	Replace first occurrence of foo with bar in ’file.txt’ in a line.	0.402987207166
Compress the file file.txt. Use level 9 compression.Keep the original file.	Forcefully compress the file myfile.txt present in this folder. Keep the original file. Show the compression ratio.	0.500847776544
Remove linuxfile folder if empty.	Command to remove linuxfile folder if empty with a log else ignore.	0.602618361588
Terminate all processes with ids 123 543 2341 3453.	End all processes with ids 123 543 2341 3453.	0.872910281554

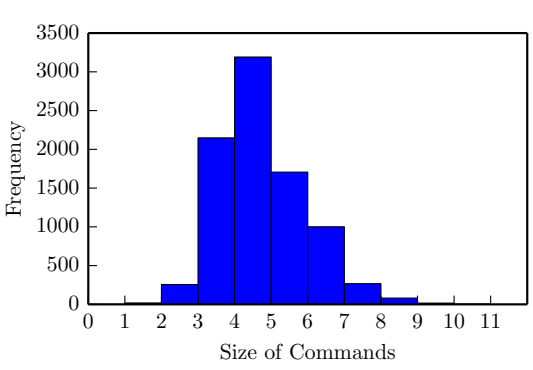
Don't consider the first 8 characters while comparing lines and display only unique lines from the file tutorial.txt on the command line along with it's number of occurences.	Consider first 8 characters and ignore case while comparing lines and display only unique lines from the file tutorial.txt on the command line along with it's number of occurences in the file.	0.905336628904

2.3.5 Length of Commands

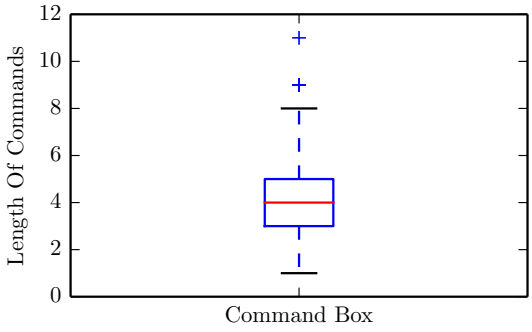
Figure 2.7(a) shows the length distribution of all Commands. Most of the commands have a range of 3-7 as show in the histogram.

Figure 2.7(b) shows a box plot of the command length distribution. This plot shows that there is only one outlier with length 11 and that the median command length is 5 at 25th percentile.

The minimum length of command is 1 and the maximum length is 11.



(a) Distribution of length of commands



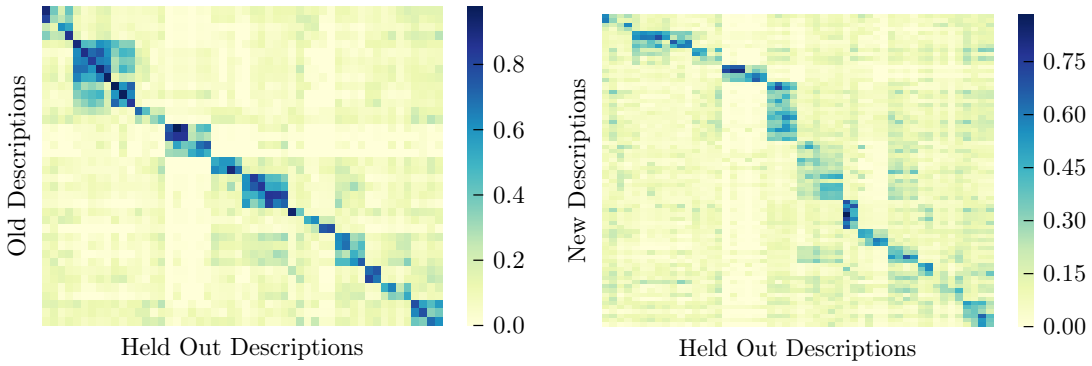
(b) Box plot of length of commands

Figure 2.7: Statistics of length of commands

2.4 Data Bias

As explained in section 2.1, a single user provides all the descriptions for a given command (for convenience in collecting the dataset). In this section we study any biases in the dataset because of this choice in the collection procedure. We collect new descriptions of 25 complete commands from a completely different set of users. We create a held out set consisting of half the original descriptions of the held out set. We compute similarities of the remaining old descriptions with the held out set and the new descriptions. In both figures, the blocks on the diagonal depict pairwise similarity values of descriptions between a single executable Linux command. The rest are similarity values between descriptions of different executable Linux commands. Each block shows the distribution of how varied the descriptions for each sample are.

Figure 2.8(a) shows high values in the diagonal, even though there are no common descriptions between both the sets. We can infer that there isn't a lot of variance in the old descriptions from the held out set. From Figure 2.8(b), we observe that similarity values of new descriptions are not very high. We can infer that for an executable command sample, the distribution of descriptions can achieve high variance and there aren't enough data points in our dataset to capture it. This leads to the conclusion that our data is biased and more variance can be achieved by incorporating more users.



(a) Heatmap of old with held out set

(b) Heatmap of new with held out set

Figure 2.8: Heatmaps to assess bias

CHAPTER 3

CLASSIFICATION USING SUPPORT VECTOR MACHINES

3.1 Introduction

In this section, we study the problem of classifying each description into it's base class, as described in 2.2.1. This exercise determines whether the main intent conveyed in the description is recognized accurately by determining it's base class. A natural language description is given as an input to a classification model and the model outputs the highest probable base class which this description belongs to. Table 2.1 shows a few examples of descriptions and their corresponding base class. We use Support Vector Machines(SVM)[20] as a learning algorithm for this classification problem.

We begin by discussing the theory behind Support Vector Machines, followed by a list of experiments conducted on the classification datasets and analyze the results obtained from the experiments.

3.2 Theory of SVM

Support Vector Machines(SVM) are a class of supervised learning models. They learn a linear classifier which maximizes the margin between the decision boundary and the nearest/most confusing data point for each class(also knows as support vectors).

Let $S = \{x_i, y_i\}_{i=1}^{m}$ be a training set with m data points and $y_i \in \{+1, -1\}$. Assume that data points are not linearly separable. We add soft constraints where a data point is given a penalty of $C\xi_i$ if it lies within the margin on the correct side of the separating hyperplane($0 \leq \xi_i \leq 1$) or if it lies on the wrong side of the hyperplane ($\xi_i \geq 1$)[21]. This

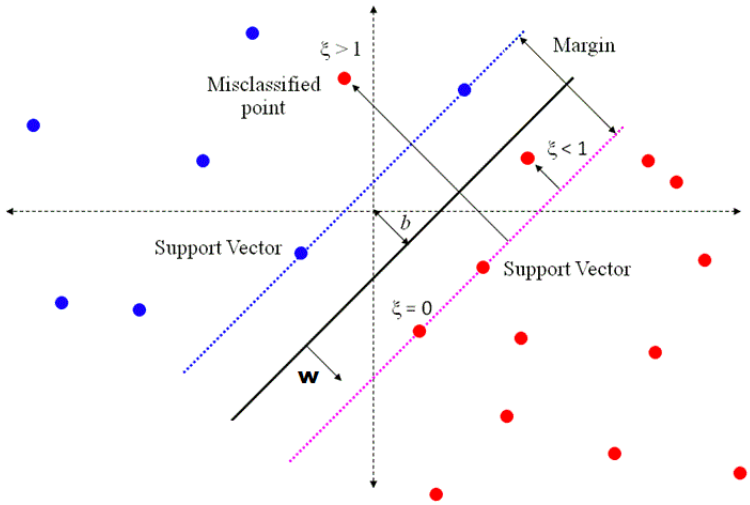


Figure 3.1: Support Vector Machine

is called *Soft margin SVM* and is represented by the following optimization problem:

$$min\ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i$$

$$\text{such that } y_i(w^T x_i + b) \geq 1 - \xi_i,\ \xi_i \geq 0$$

The points with minimal margin are closest to the decision boundary and are called *Support Vectors*. C is a regularization parameter with L1 regularization. A small value of C will indicate linear separability of data. Figure 3.1 shows binary classification of 2-dimensional data[22]. The minimum margin from the decision boundary is $\frac{1}{\|w\|}$ and the support vectors for each class lie on the lines parallel to the margin belonging to it's corresponding side.

3.2.1 Multi-class SVM

So far, we have discussed Binary classification. Let's assume we have K classes. We can extend SVMs to perform multi-class classification using the following techniques:

- *One-versus-all*: Train K one-versus-all classifiers and choose the class with the highest score.
- *One-versus-one*: Train $\frac{K(\ K-1)}{2}$ classifiers which includes all unique unordered pairs

of K classes. Choose the class most obtained among all results.

- *DAGSVM*: This is similar to One-versus-one in the training phase as it creates $\frac{K(\ K-1)}{2}$ classifiers. However, during the testing phase, it uses a rooted binary directed acyclic graph which has $\frac{K(\ K-1)}{2}$ internal nodes and K leaves. Each internal node is a binary SVM. First, root node is used for classification and then depending on the result, the an appropriate path is taken till a leaf node is reached. This leaf node is the prediction for the test data point[23].

For our purposes, we choose One-versus-one strategy for multi-class classification.

3.3 Metrics

This section shows the following metrics for each experiment:

- *Overall Precision*: This metric calculates the global precision metric by counting the total true positives, false negatives and false positives.

$$\text{Overall Precision} = \frac{\sum_{i=1}^{C} \text{True Positives of class}_\text{i}}{\sum_{i=1}^{C} \text{(True positives+False Positives) of class}_\text{i}}$$

- *Mean Precision*: This value calculates the precision metric for each class and finds their unweighted mean.

$$\text{Mean Precision} = \frac{1}{C} * \sum_{i=1}^{C} \frac{\text{True positives of class}_\text{i}}{\text{(True positives + False positives) of class}_\text{i}}$$

where C is the number of classes.

- *Mean Weighted Precision(MWP)*: This value is a weighted mean of precision score

of each class, based on the class distribution in the data.

$$\text{MWP} = \sum_{i=1}^{C} \frac{k_i}{C} * \frac{\text{True positives of class}_\text{i}}{\text{(True positives + False positives) of class}_\text{i}}$$

where C is the number of classes, k_i is the number of instances of class$_\text{i}$

We use Mean Weighted Precision as our main metric as it takes into account precision score for each class which measures individual class performance and the weighted average combination represents the performance globally taking the class distribution into account.

3.4 Experiments and Results

As mentioned in section 3.1, we take a description as an input and classify which Linux command it belongs to. We perform multi-class classification using SVM for the two datasets detailed in Chapter 2: CL-21 and CL-42. We create a Bag-of-Words(BoW) representation of an input description. Two representations are created for each description using NLTK[24] tokenizer and an in-built tokenizer in the python module *sklearn*[25]. We experiment to find optimal parameter c and report the results in the form of a confusion matrix.

3.4.1 Experiments on CL_21

We take all the sentences belonging to this dataset and remove the ones which contain their base class name verbatim. Table 2.3 contains a few examples for such descriptions. This check is necessary to preserve the integrity of classification, as the base class name in the sentences would give away class information to the model, leading to very high accuracy. After filtering, the number of (Command, Description) pairs reduce from 2955 to 2916.

Next we use NLTK and sklearn tokenizers to construct the BoW representations of the input descriptions and compare their performance across different tokenizers for the linear SVM. The size of NLTK vocabulary is 1209 and the size of sklearn vocabulary is 1020.

Table 3.1: Metrics for CL-21 using sklearn vocabulary

Data	**Overall Precision**	**Mean Precision**	**MWP**
Train	0.851	0.906	0.961
Test	0.854	0.832	0.962

Table 3.2: Metrics for CL-21 using NLTK vocabulary

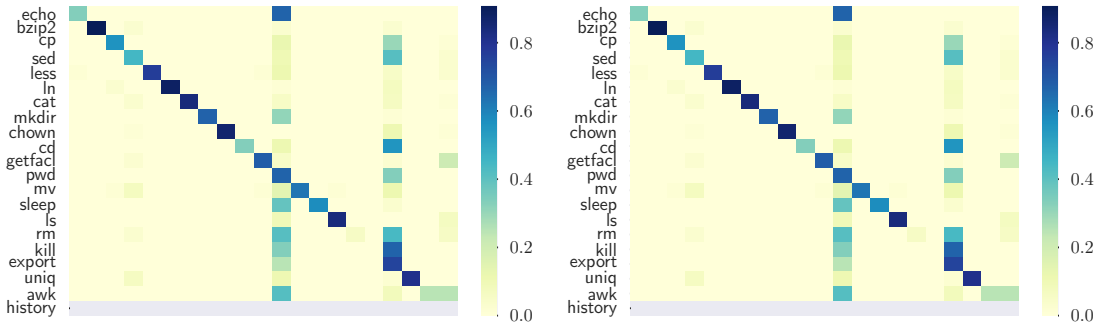
Data	**Overall Precision**	**Mean Precision**	**MWP**
Train	0.818	0.849	0.957
Test	0.845	0.779	0.951

Tables 3.1 and 3.2 show the best metric values obtained from our experiments.

From our experiments, we note that best test Overall Precision using sklearn vocabulary is achieved at training size of 0.9 with the regularization parameter c = 0.85. For sklearn vocabulary, the best test MWP is achieved at a regularization parameter c = 0.8 after training on 70% of data.

The best test Overall Precision using NLTK vocabulary is achieved at training size of 0.8 with the regularization parameter c = 0.8. For NLTK vocabulary, the best test MWP is achieved at a regularization parameter c = 0.95 after training on 70% of data.

Comparing data from Table 3.1 and 3.2, we observe that test MWP with sklearn vocabulary are higher than NLTK vocabulary. It is interesting that there was a difference of 1% in test MWP when different tokenization mechanisms were used. sklearn is a better tokenizer for this dataset than NLTK with a smaller vocabulary size.



(a) Test set: sklearn

(b) Test set: NLTK

Figure 3.2: Confusion matrices of Test Data: CL-21

3.4.2 Experiments on CL-42

We applied the same filters as before and the number of (Command, Description) pairs reduce from 7560 to 7496. Again, we construct BoW representation using NLTK and sklearn vocabularies. We create a vocabulary using tokenizer of *nltk* library and create Bag of Words representation of all descriptions. The size of the NLTK vocabulary is 1996 and the size of sklearn vocabulary is 1692. We experiment with different values of c parameter to determine the best value. Table 3.3 and Table 3.4 show the best metrics throughout training across different c values and training set sizes.

Table 3.3: Metrics for CL-42 using sklearn vocabulary

Data	**Overall Precision**	**Mean Precision**	**MAP**
Train	0.825	0.866	0.946
Test	0.842	0.811	0.939

Table 3.4: Metrics for CL-42 using NLTK vocabulary

Data	**Overall Precision**	**Mean Precision**	**MWP**
Train	0.818	0.849	0.957
Test	0.845	0.779	0.951

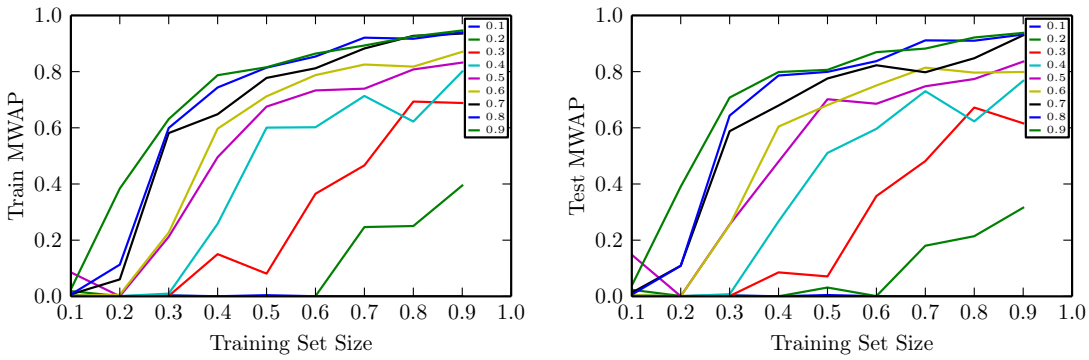
From Figure 3.3 and 3.4, we observe that the best test MWP using sklearn vocabulary is achieved at training size of 0.8 with the regularization parameter c = 0.85. The best test accuracy using nltk vocabulary is achieved at training size of 0.8 with the regularization parameter c = 0.95.

For sklearn vocabulary, the best test MWP is achieved at a regularization parameter c = 0.85 after training on 70% of data. For NLTK vocabulary, the best test MWP is achieved at a regularization parameter c = 0.95 after training on 70% of data.

From Table 3.3 and 3.4, we observe that test MWP with NLTK vocabulary are higher than sklearn vocabulary. It is interesting that there was a difference of 2% in test MWP values. NLTK has a larger vocabulary size and for a larger dataset, having a larger vocabulary size works at an advantage.

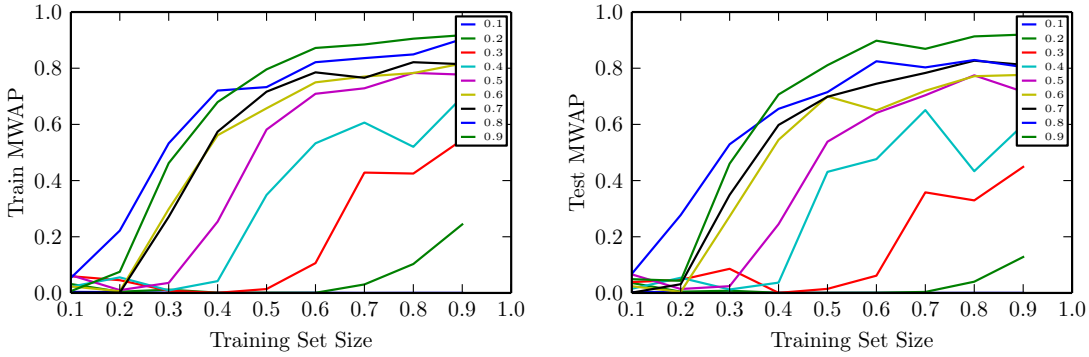
Figures 3.3 ad 3.4 show the progress of MWP across different training set sizes for different values of c parameter. For sklearn vocabulary, c value of 0.9 does consistently better throughout training than when the peak is reached at 0.8 for both train and test sets. For other values of c, the progress across training set sizes is very less. For NLTK vocabulary, value of 0.9 does a lot better after training set sizes of 0.3. The high values of c indicate that data is not linearly separable.

Figure 3.5 shows confusion matrices of test predictions for both sklearn and NLTK Vocabulary.



(a) Train set (b) Test set

Figure 3.3: Training graphs for optimal c value: sklearn

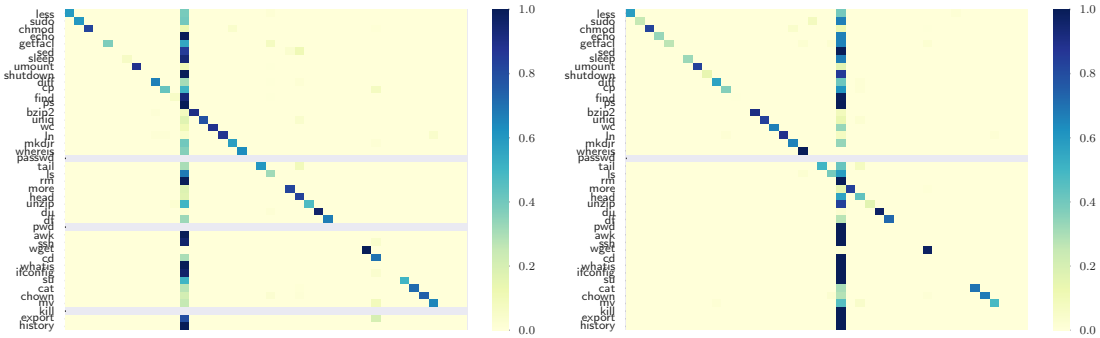


(a) Train set (b) Test set

Figure 3.4: Training graphs for optimal c value: nltk

For sklearn, it's interesting to note that 3 classes: passwd, pwd and kill are not being predicted at all. During 70-30% split of data for training, datapoints of passwd are not present during training which explains why this class is not predicted. Both kill and pwd classes contain only 5 training data points, which is very less data for a class to train and predict on. A lot of incorrect predictions are classified as ps. ps class contains 22 natural language descriptions with a maximum size 15. These descriptions contain almost an equal proportion of proportion of proper nouns and generic English words such as show, display, process, state, user etc. Such words are present in descriptions of many other commands. Less and generic words maybe the reason why many classes get classified as ps.

For NLTK vocabulary, the class passwd is not being predicted at all. Contrary to split for sklearn, one data point of this class was present during training. The class ssh is being confused with other classes. In fact, on multiple runs with same parameters, the classes echo, ps, awk, rm get confused. The classes echo, ps, awk contain many proper nouns. This maybe the reason classification is tough for these classes.



(a) Test set: sklearn

(b) Test set: NLTK

Figure 3.5: Confusion matrices of Test Data: CL-42

3.5 Results using new dataset

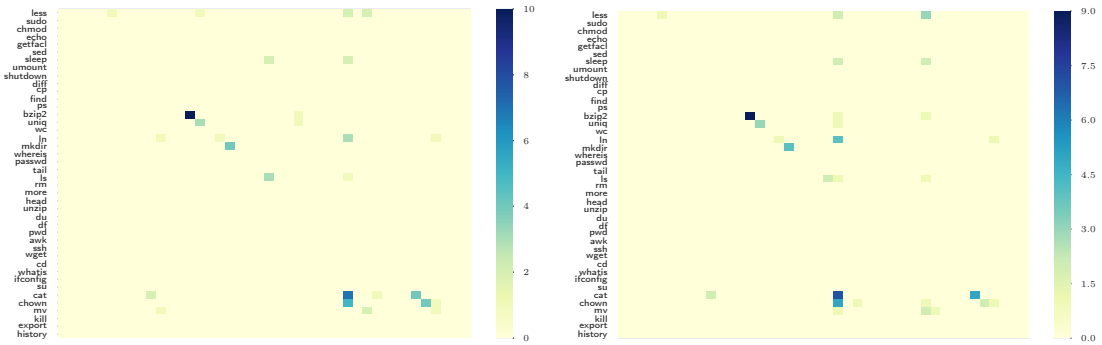
As mentioned in section 2.6, we collect new descriptions to assess the bias present in the dataset. We create a BoW representation of the new data using both sklearn and NLTK vocabulary. We train on 70% of CL-42 dataset, use the new descriptions as test set and report the results. As the new data obtained contains classes mentioned in Table 2.3, we removed those descriptions. The total of 67 descriptions remained in the test set.

Table 3.5 shows the mterics of the new test set using both vocabularies and parameters used. It's interesting to note that the values for all metrics are exactly the same. This is not the case with our previous results.

Table 3.5: Metrics for new test set

Vocabulary	c	Overall Precision	Mean Precision	MWP
sklearn	0.8	0.44776119403	0.393137254902	0.772139303483
nltk	0.95	0.44776119403	0.393137254902	0.772139303483

The confusion matrices of Figure 3.6 show the difference in predictions between the two classifiers. Both classifiers perform equally well in predicting correct classes, which can be seen in the diagonal of both figures. The classifier trained on sklearn gets the most confused with ”awk” class. As mentioned earlier, this class contains very less descriptions and most of the arguments of this class are proper nouns. This can be one of the reasons other classes may generalize to this class. For NLTK Vocabulary, the most false predictions are for rm and wget class. rm contains very less data and hence the same reason can apply here.



(a) New Test set Predictions: sklearn (b) New Test set Predictions: NLTK

Figure 3.6: Confusion matrices of New Test Data

CHAPTER 4

TRANSLATION WITH SEQUENCE TO SEQUENCE NEURAL NETWORKS

4.1 Introduction

Machine Translation is automatic translation of one language to another using examples of human produced translations. There are two categories of Machine Leaning models:

- *Statistical Machine Translation*: Translations are based on statistical models and their parameters are derived from corpora of bilingual text.

$$\hat{E} = \underset{E}{\operatorname{argmax}}\, Pr(E|F;\theta)$$

where E is the target sentence, F is the source sentence,

θ are the parameters of the model.

There are many types of models based on the entity being translated: Word-based translation, Phrase-based translation, Syntax-based translation etc.

- *Neural Machine Translation*:A class of sequence to sequence[15] neural machine translation approaches has been proposed recently which consists of an encoder and a decoder. The encoder creates an embedding of the input sentence capturing it's meaning and the decoder produces the translation using this hidden state. An attention mechanism[26] determines context by capturing association between the current decoder state and all input words in order to generate the next output embedding in translation. Many enhancements have been made to this model such as using bi-directional Recurrent Neural Networks to make sure we don't have catastrophic forgetting during translation (assuming the spread of concepts in both the languages is similar).

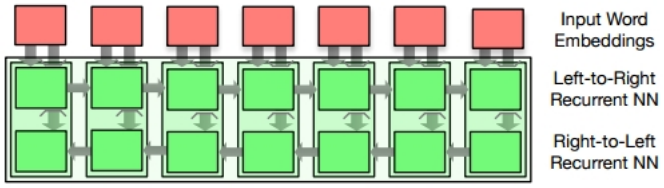


Figure 4.1: Encoder Architecture

4.2 Sequence to Sequence Networks

Recurrent Neural Networks (RNN) and Long Short Term Memory Networks (LSTM) are used to solve sequence to sequence mapping problems. Encoder-Decoder architectures along with an explicit alignment mechanisms are used. They are briefly described below and the figures are taken from [27]

Encoder

An Encoder produces a vector representation of fixed dimension for an input sentence. During the encoding phase, an embedding lookup of the input word and then maps the word along with it's left context into a hidden state. A bi-directional LSTM performs the same procedure in the opposite direction capturing the context towards the right[27]. Figure 4.1 depicts an Encoder Architecture.

Decoder

Decoder conditioned on the input sequence produces a target sentence. It takes the embedding of the previous output word generated, previous hidden state and current context and generates a new decoder hidden state and an output prediction. Figure 4.2 depicts a Decoder Architecture.

Bi-Directional Encoder

In a bi-directional encoder, two encoders are used: one travels forward in an input sequence and the other travels backward. The final hidden states from both are combined and fed into the decoder. A reverse encoder helps reduce the length of dependencies between subset of words which gradually helps learn longer dependencies for large sentences. It is useful for pair of languages containing similar sentence structures.

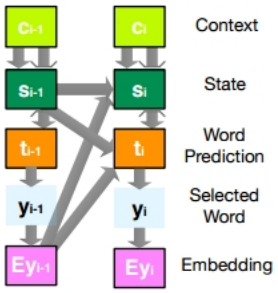


Figure 4.2: Decoder Architecture

4.3 Attention Mechanism

An attention mechanism[27] creates an association between the current decoder state and all the input words. A feedforward network is used to calculate weighted attention value for all input words to represent it's relevance to the current output word being generated. We use the model provided by Bahdanau et. al [26].

4.4 Optimization of Neural Network

Training Neural Network requires minimizing an objective function which denotes loss w.r.t. parameters of the network. Gradient based search techniques such as back propagation are used for training Neural Networks. In this work, we experiment with two techniques which are described below.

Stochastic Gradient Descent

A standard Gradient Descent Technique minimizes parameter θ of an objective function $J(\theta)$.

$$\theta = \theta - \alpha \nabla_\theta E[J(\theta)]$$

Figure 4.3 shows a 1-dimensional example of gradient descent with x_0 as an initial value and x_5 being the minimum of the optimization function $L(w)$. Stochastic Gradient Descent updates parameters using only a few training examples or a mini-batch as opposed to Batch

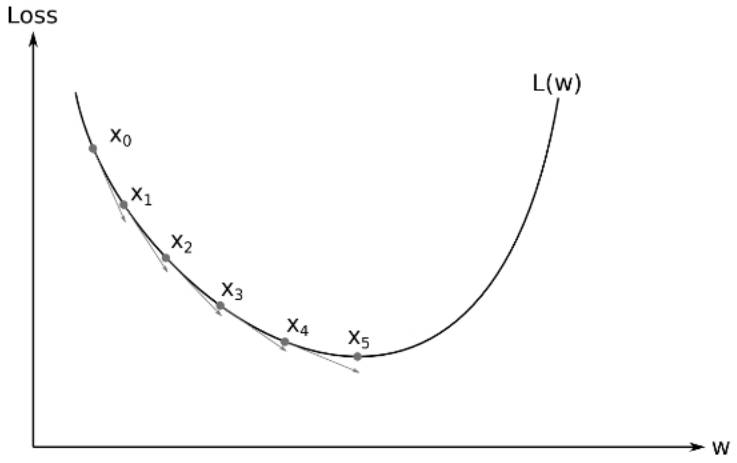


Figure 4.3: Gradient Descent

Gradient Descent which used all training examples to compute one update. It is faster due to less amount of computation in one iteration. It produces updates with high variance and causes high fluctuation in the objective function. It's important for the data to be randomly shuffled for each update, otherwise it can lead to poor convergence.[28] provides an overview of various optimization algorithms.

Adam Optimization

Adaptive Moment Estimation [29] is another method used to estimate parameters for non-convex optimization functions in Deep Learning. It stores exponentially decaying average of past gradients m_t and exponentially decaying average of past squared gradients v_t.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

where m_t is first moment, v_t is second moment of gradients,$\beta_1, \beta_2 \in [0, 1)$ are exponentially decaying rates for moment estimates. As these values are biased towards 0 in the initial steps, bias-corrected estimates $\hat{m}_t$ and $\hat{v}_t$ are used. The update rule of Adam opti-

mizer is the following:

$$\hat{\theta}_t = \hat{\theta_{t-1}} - \frac{\eta}{\sqrt{\hat{v_{t-1}}} + \epsilon} \hat{m_{t-1}}$$

Beam Search

The simplest way to get a translation is *Greedy Best Search* by choosing the word with highest probability as the next word. Instead of choosing the best hypotheses at every time-step, we choose bw (beam width) best hypotheses with the highest probability conditioned on the bw hypotheses selected in the previous time step. This enables us to choose a translation with the highest overall probability. A higher beam width depicts better translation quality. This strategy works very well on long sentences.

4.5 Metrics Used

We use the following metrics to measure the translation quality. Let N be the total number of commands and L is the length of the command.

- *BLEU and Modified N-gram Precision*: Bilingual Evaluation Understudy (BLEU) score[30] is a popular metric to measure the translation quality. Central idea behind BLEU is: *The closer a machine translation is to a professional translation, the better it is.* The measure of closeness can be defined as the fraction of n-grams of all candidate translations in the corpus common with n-grams of their corresponding reference translations in the corpus.

$$\text{Modified N-gram Precision} = \frac{\sum_{C \in \text{Candidates}} \sum_{\text{n-gram} \in C} \text{Count}_{clip}(\text{n-gram})}{\sum_{C' \in \text{Candidates}'} \sum_{\textit{n-gram}' \in C'} \text{Count}(\text{n-gram}')}$$

  A modified uni-gram precision measures the *accuracy* of a translation corpus by checking the fraction of words of reference corpus present in the translation corpus, clip the total count of each candidate word by its maximum reference count, add these

clipped counts and divide by the total number of candidate words. With a higher value of n, precision of variable length phrases will be measured. This precision would provide a measure of *fluency* of all translations denoting it's capacity to "learn" the target language and it's grammar. We calculate a weighted sum using n-grams and penalize the score with Brevity Penalty(BP) accounting for the mismatch of length between reference and the candidate.

$$\text{BP} = \begin{cases} 1 & \text{if c} > \text{r} \\ \exp(1 - \frac{r}{c}) & \text{if c} \leq \text{r} \end{cases} \qquad \text{BLEU} = \text{BP} * \exp(\sum_{i=1}^{N} w_i log(p_i))$$

From Figure 2.7(b), we observe that 75 percentile of commands lengths are below 7. Hence, we choose to calculate BLEU score with 7 as the maximum size of n-gram. This would provide us a good metric to judge the corpus.

- *Word Accuracy*: This metric calculates the accuracy of each command by calculating correctly placed words in the translation and normalizing by the number of commands.

$$\text{Word Accuracy} = \frac{\sum_{i=1}^{N} \sum_{l=1}^{L} \text{translation}_i^l == \text{reference}_i^l}{N}$$

where translation_i^l is the l-th word of i-th translation, candidate_i^l is the l-th word of i-th translation.

This metric depicts the ability of the model to learn the syntax of commands as it takes into consideration proper placement of correct words.

- *Total Accuracy*: This metric measures the number of commands which are translated verbatim.

$$\text{Total Accuracy} = \frac{\text{Count(translation} == \text{candidate)}}{\text{total number of translations}}$$

This metric denotes the capacity of the model to generate an executable command.

- *Command Accuracy*: Syntax of an executable Linux consists of 3 parts: A Linux Command, it's options and arguments. We calculate accuracy of each part by dividing the reference and translation command into these 3 syntax parts and calculate the number of matches for each part. Command accuracy is an equally weighted sum of the 3 values calculated.

$$\textit{Command Accuracy} = \frac{\text{Linux Command Accuracy} + \text{Options Accuracy} + \text{Arguments Accuracy}}{3}$$

4.6 Experiment Map for Small Datasets

We run a Sequence to Sequence network on the datasets mentioned in Section 2.2.2 using Tensorflow API [31]. We first experiment on small datasets namely COMPOSITIONAL-40 and IID-40 using various parameters and report results. We then use the most optimal parameters on the complete dataset. We perform the set of experiments shown in Table 4.1 for COMPOSITIONAL-40 and IID-40.

Table 4.2 shows the default parameters of the model.

Table 4.1: Experiment Set for 40%

Experiment ID	**Description**
Experiment I	Choose learning rate of Stochastic Gradient Descent among 0.01,0.1,0.5,1.0 for Uni-Directional encoder network.
Experiment II	Choose learning rate of Stochastic Gradient Descent among 0.01,0.1,0.5,1.0 for Bi-Directional encoder network.
Experiment III	Run Attention Model on the best Encoder model from I and II.
Experiment IV	Experiment with learning rate of Adam optimizer among 0.01, 0.001 from the best model till Experiment III.
Experiment V	Choose the best beam width from 5 (default), 6, 8, 10.

Table 4.2: Default Parameters

Parameters	Values	Parameters	Values
num_units	32	num_layers	2
encoder_type	unidirectional	attention	None
optimizer	Stochastic GD	Learning Rate	1.0
decay_factor	1.0	num_train_steps	12000
unit_type	lstm	dropout rate	0.2
source_reverse	False	beam_width	0

We show different metrics for all experiments and perform comparisons based on training loss, validation perplexity, command accuracy, word accuracy, BLEU score, N-gram Distribution and command distribution.

4.7 Experiments and Results for IID-40

We perform the experiments listed in Table 4.1 for the IID-40 dataset and show the results. We show trends of training loss, development perplexity, development accuracy, testing accuracy.

4.7.1 Experiment I and II

As mentioned in Table 4.1, we experiment with two models: one containing uni-directional encoder and the other containing bi-directional encoder. We use stochastic gradient descent as an optimizer and experiment with 4 learning rates for each model: 0.01, 0.1, 0.5, 1.0.

Performance on Development set

Table 4.3 and 4.4 show the best metrics obtained from both models. Table 4.3 shows that no one learning rate does consistently well in all tasks. However, we see that 0.5 and 1.0 perform almost equally well in across all metrics. We choose 1.0 as it's values of best validation accuracy, best validation word accuracy and best validation command accuracy are very close to the highest values among all values.

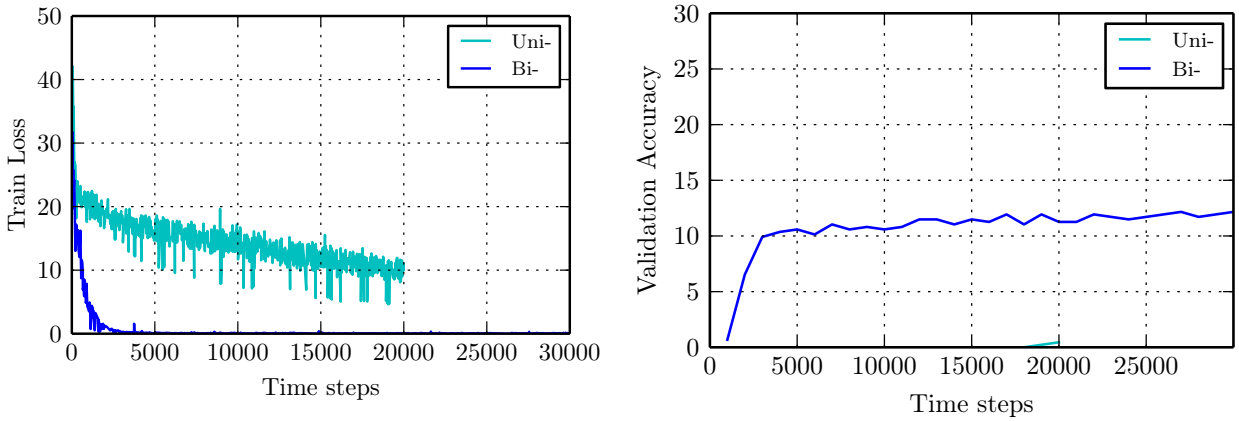
Table 4.3: Best Metric values for uni-directional encoder: IID-40

Learning Rate	Best Development BLEU-7 Score	Best Validation Accuracy	Best Validation Word Accuracy	Best Validation Command Accuracy
0.01	0.0	0.5	30.6	0.6
0.1	25.2	10.4	**53.8**	**0.7**
0.5	**27.8**	11.3	53.0	**0.7**
1.0	26.3	**11.9**	53.7	**0.7**

Table 4.4: Best Metric values for bi-directional encoder: IID-40

Learning Rate	Best Development BLEU-7 Score	Best Validation Accuracy	Best Validation Word Accuracy	Best Validation Command Accuracy
0.01	0.0	0.0	24.2	0.5
0.1	21.7	10.6	53.7	**0.7**
0.5	24.8	10.4	53.2	**0.7**
1.0	**28.1**	**12.2**	**55.5**	**0.7**

Table 4.4 shows that learning rate 1.0 gives the highest metric values.



(a) Training Loss

(b) Validation Accuracy

Figure 4.4: Training Parameters with learning rate 1.0: IID-40

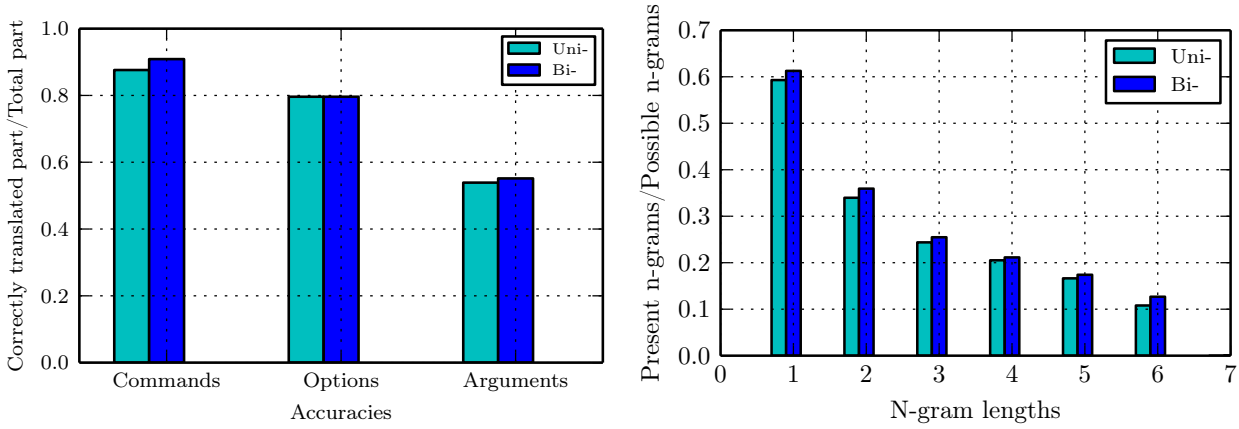
Figure 4.4 shows a performance comparison of these two models with their optimal learning rates, 1.0 for both models. We can observe that for a bi-directional model, the training loss decreases at a faster rate than uni-directional model. The development accu-

racy for a bi-directional model increases steeply till 4000 time steps and there is a gradual decrease after that. On the other hand, development for a uni-directional accuracy doesn't increase till 18,000 time steps.

Performance on Test Set

Figure 4.5 shows the how well n-gram and command distributions are learnt by the test set for the best development BLEU-7 score and for the best development command accuracy for both encoders.

In both cases, a bi-directional encoder performs better on the test set. Neither models learn n-grams of size 7 and for learning n-grams of size 6, bi-directional performs 1.8% better than a uni directional encoder. In terms of learning commands, it performs 3.2% better and in terms of learning arguments of a command, it performs 1.25% better.



(a) Command Distribution

(b) N-gram Distribution

Figure 4.5: Distributions of Test set with uni- and bi- directional encoders: IID-40

Results of Test set

Tables 4.5, 4.6 and 4.7 show a few translations of the obtained by a bi-directional encoder. The best model produces 116 verbatim translations of corresponding reference commands out of 888, 112 of which are unique translations. Verbatim translations generate the exact

copy of the executable reference command. There are 55 accurate translations of command mount, 22 of command umount, 17 of chmod and the rest belonging to different commands.

Table 4.5: Verbatim translations with bi-directional encoder model: IID-40

Input Description	**Correct Translation**
Mount the file systems of the given types described in fstab read/write without writing in /etc/mtab. In case of a loop mount with encryption, read the passphrase from file descriptor num instead of from the terminal.	mount -n -p -w -a
Move to next page of command line. Show the contents of myfile.txt. Display prompt instead of ringing bell when an illegal key is pressed. Set screen lines to use as '10'.	more -pd -10 myfile.txt
Delay for 2 seconds, 11 minutes and 15 minutes.	sleep 2s 11m 15m

Table 4.6: Almost correct translations with bi-directional encoder model: IID-40

Input Description	**Reference**	**Output**
Give execute and read permissions for all the content in the directory foldername to all users. Generate a log.	chmod -Rv ugo=rx foldername/	chmod ⟨**unk**⟩ ugo=rx foldername/
Show the differences between one.txt and two.txt files in RCS format. Do not include blank lines in difference.	diff -nB one.txt two.txt	diff ⟨**unk**⟩ one.txt two.txt
Pause command line for 1 hour 15 seconds.	sleep 1h 15s	sleep 1h **2m**

Table 4.6 shows a few commands where most of the command is translated verbatim

and only the options are not translated. Table 4.7 shows a few translations where even the base class is not translated correctly.

Table 4.7: Completely Incorrect Translations with bi-directional encoder Model: IID-40

Input Description	**Reference**	**Output with bi-directional encoder**
Allows the usage of function function˙name to all the child processes.	export -f function˙name	mount ⟨unk⟩ -r /mydata
Create a copy of instructions.txt named backup. Copy only if backup not present in current folder.	cp -n instructions.txt backup	diff ⟨unk⟩ ⟨unk⟩ ⟨unk⟩
Get one line description of head command.	whatis head	cat ⟨unk⟩

4.7.2 Experiment II and III

As mentioned in Table 4.1, we experiment with two models: bi-directional encoder with and without attention. The attention model used is described in section 4.3. We use stochastic gradient descent as an optimizer with learning rate 1.0.

Performance on Development set

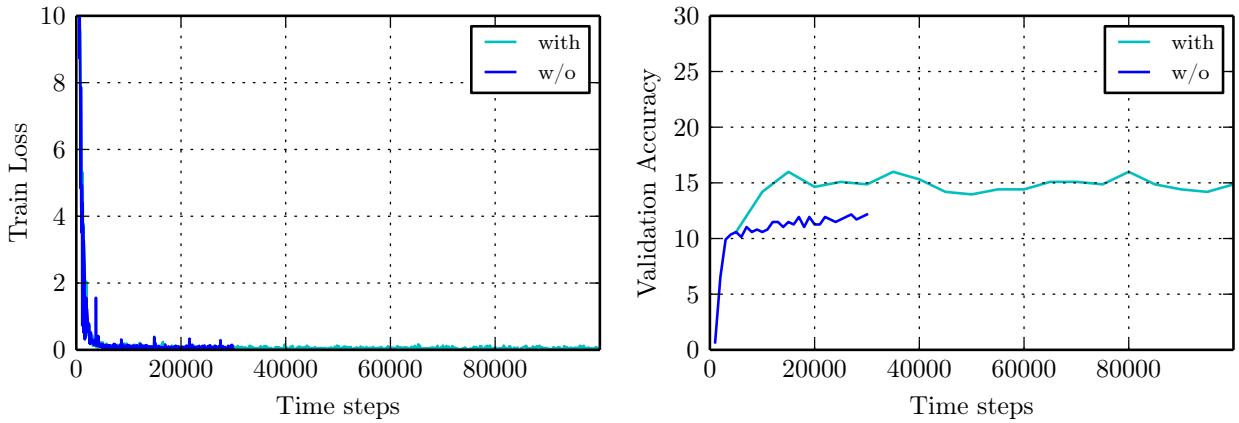
Table 4.8 shows the best metrics of bi-directional encoder model with and without attention. We have run the model with attention for 80,000 time steps and the model without attention for 30,000 time steps. We see that the attention model has consistently higher values for all metrics.

Figure 4.6 shows a performance comparison of the two bi-directional models. For training loss, both models show the same trend: a steep fall at first and then gradual decrease. The validation accuracy shows interesting trends. The attention model grows similar to the one

Table 4.8: Best Metric values with Bi-directional Encoder Models: IID-40

Attention type	Best Development BLEU-7 Score	Best Validation Accuracy	Best Validation Word Accuracy	Best Validation Command Accuracy
without	28.1	12.2	55.5	0.7
with	**32.3**	**16.0**	**59.7**	**0.8**

without till 5000 iterations and then rises sharply. It's value is 3-5% higher for all iterations.



(a) Training Loss

(b) Validation Accuracy

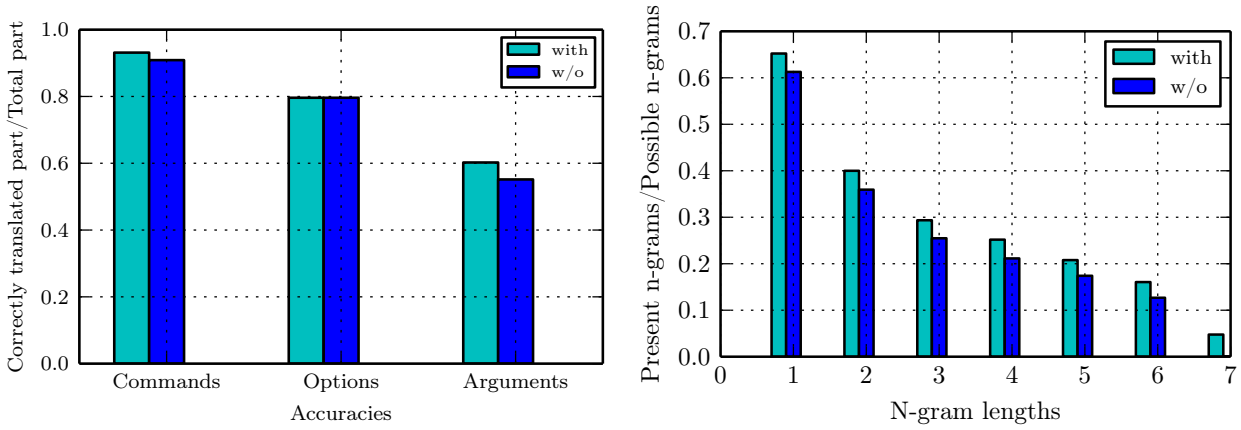
Figure 4.6: Training Parameters with for bi-directional encoders: IID-40

Performance on Test Set

Figure 4.7 shows the how well n-gram and command distributions are learnt by the test set for the best development BLEU-7 score and for the best development command accuracy.

Both distributions depict that the attention model performs better on the test set. Only the attention model learns a 7-gram command. For n-grams of size 6, attention model performs 5% better, 2.25 % for learning commands and for learning arguments of a command, it performs 5% better. It learns options equally well as the model without attention. It's interesting to note that the maximum length of translation learned by the

attention model is 8 while the model without attention learns a command size of 6. It can be due to the fact that the arguments learned are better in the attention model.



(a) Command Distribution

(b) N-gram Distribution

Figure 4.7: Distributions of Test set for bi-directional encoders: IID-40

Results of Test set

Tables 4.9, 4.10 and 4.11 show a few translations of the obtained by a bi-directional encoder with attention model. The best model produces 151 verbatim translations of corresponding reference commands out of 888, 143 of which are unique translations. There are 52 accurate translations of command mount, 25 of command umount, 25 of chmod, 10 of sleep and the rest belonging to different commands.

Table 4.10 compares translations of models with and without attention. We briefly describe which model translates which part of the syntax better for the same input description. The first 4 rows show the translations which attention model learns more accurately. Translation 4 shows that neither model can learn the regex of awk command properly, but the attention model learns the argument ”table.csv”. The last three translations depict where the model with attention performs better. It learns the main command ”sudo” while the model with attention learns ”chmod”. It learns a part of regex ”hello” while the model with attention learns ⟨unk⟩. It learns the correct command ”chown” while the model with

Table 4.9: Verbatim translations using bi-directional encoder + Attention: IID-40

Input Description	Translation
Move to next page of command line. Show the contents of myfile.txt. Display prompt instead of ringing bell when an illegal key is pressed. Set screen lines to use as '10'. more -pd -10 myfile.txt	mount -v -l -n -t test
Display all the files in current directory with access time newer than modification time of test.txt file.	find . -anewer test.txt
Show the content of instructions.txt with $ at end of each line, non-printing characters and TAB character as ^I.	cat -te instructions.txt

attention learns "tar".

Table 4.11 shows the commands which are completely inaccurate. IT can be noted that the attention model gets "chown" and "tar" confused. It's tough to say why that can be happening as their functions are entirely different: chown is for changing file ownership and tar is for compressing files into an archive. It's difficult for the model to translation commands like export, awk etc. as the command consists of mostly proper nouns and regular expressions, which requires a lot of data to learn. Neither of these classes have enough commands or descriptions to learn complicated expressions.

4.7.3 Experiment III and IV

As mentioned in Table 4.1, we experiment with Adam optimizer using different learning rates: 0.01 and 0.001 on a bi-directional model with attention.

Table 4.10: Translations of bi-directional encoder with and without attention: IID-40

Input Description	**Reference Translation**	**Translation with attention**	**Translation without attention**
Order the contents of random.txt ignoring starting blanks and in reverse dictionary order. For future retrieval of output use sorted.txt to save it.	sort -brd random.txt -o sorted.txt	sort -r random.txt -o sorted.txt	sort -nk2,2 nums.txt -o /anthrax.txt
Show the archive test.tar contents.	tar -tf test.tar	tar -tf test.tar	rm linuxfile.log
Move bleach.txt -⟩ anime/bleach.txt and do not over-write. Set SEL security context to default and show the jobs completed.	mv -nvZ bleach.txt anime/	mv ⟨unk⟩ ⟨unk⟩ bleach.txt anime/	unzip ⟨unk⟩ ⟨unk⟩ -d ⟨unk⟩ ⟨unk⟩
Print column 1 and 2 contents of table.csv with ',' as a field separator.	awk 'OFS=",";print 1,2' table.csv	awk ⟨unk⟩ ⟨unk⟩ ⟨unk⟩ ⟨unk⟩ table.csv	awk ⟨unk⟩ print ⟨unk⟩ ⟨unk⟩ ⟨unk⟩ marks.txt
Invalidate timestamp of current user.	sudo -k	chmod ⟨unk⟩ ananya	sudo ⟨unk⟩
Open list.txt.Highlight all locations that match the "hello world" word. Consider small and capital letters as identical.	less -Ip "hello world" list.txt	less ⟨unk⟩ ⟨unk⟩ list.txt	less ⟨unk⟩ hello list.txt
Make myself as the owner of file1.txt and file2.txt present in this folder without any error messages.	chown -f myself file1.txt file2.txt	tar ⟨unk⟩ 755 directory-name/	chown ⟨unk⟩ ⟨unk⟩ ⟨unk⟩ tmp

Table 4.11: Incorrect translations of attention model: IID-40

Input Description	Reference	Translation with attention
Allows the usage of function function˙name to all the child processes.	export -f function˙name	getfacl -h
Make myself as the owner of file1.txt and file2.txt present in this folder without any error messages.	chown -f myself file1.txt file2.txt	tar ⟨unk⟩ 755 directory-name/
Delete onepiece.txt from manga.tar archive.	tar –delete -f manga.tar ”onepiece.txt”	pwd
Print contents of marks.txt.	awk ’print’ marks.txt	diff -si chk1.txt chk2.txt

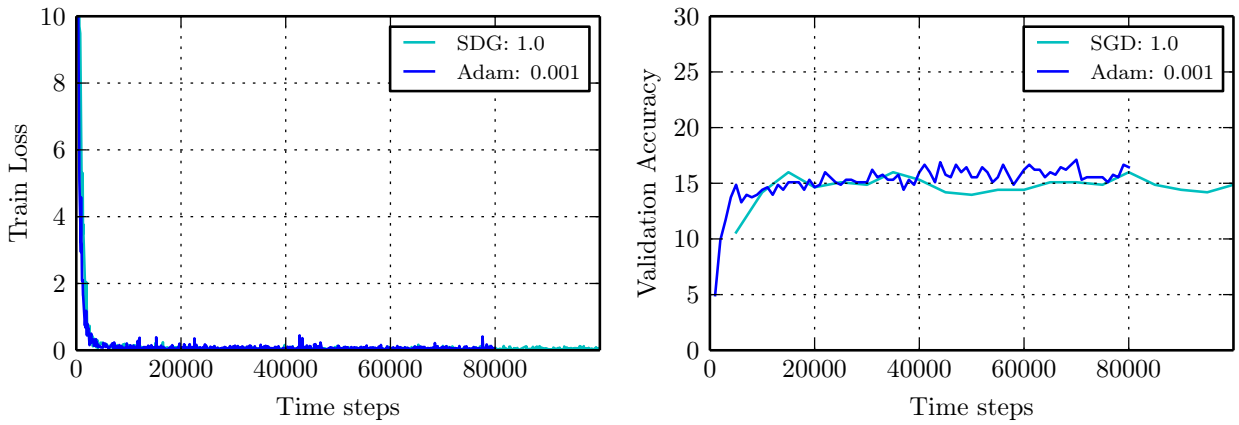
Table 4.12 shows the metrics obtained using both learning rates of Adam optimizer. In terms of BLEU score, 0.01 performs better and in terms of all other accuracies, 0.001 performs better. We choose 0.001 learning rate as it performs better in 3/4 metrics.

Table 4.12: Best Metric values using Adam optimizer: IID-40

Learning Rate	Best Development BLEU-7 Score	Best Validation Accuracy	Best Validation Word Accuracy	Best Validation Command Accuracy
0.01	**35.3**	15.1	61.0	0.8
0.001	33.6	**17.1**	**61.6**	**0.8**

We compare training of models Adam optimizer with learning rate 0.001 and stochastic gradient descent with learning rate 1.0.

Figure 4.8 show trends of training loss and development accuracy for both models. Training loss trends are very similar for both optimizers. Validation accuracy for both optimizers show similar growing trends till 40,000 time steps. Model with Adam optimizer has a higher accuracy after 40,000 time steps even though it’s trend is erratic.



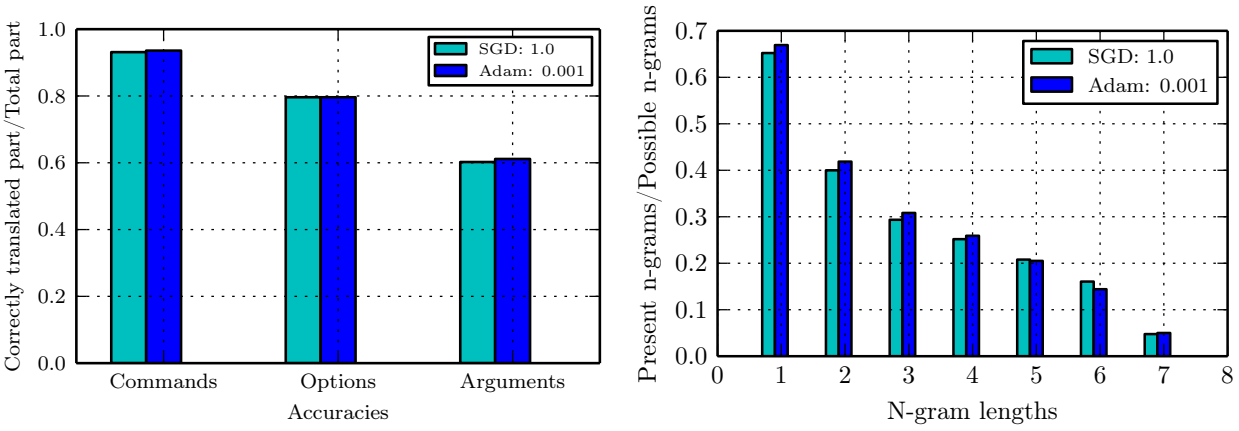
(a) Training Loss

(b) Validation Accuracy

Figure 4.8: Training Parameters with SGD and Adam optimizers: IID-40

Performance on Test set

Figure 4.9 shows comparison of n-gram and command distributions between model using Adam optimizer with learning rate 0.001 with the model using stochastic gradient descent with learning rate 1.0. Both models learn n-grams of size 7 with the model using Adam optimizer performing 0.2% better. Adam optimizer performs 0.45% better for learning Linux commands and 0.9% better for learning arguments. They both learn options equally well.



(a) Command Distribution

(b) N-gram Distribution

Figure 4.9: Distributions of Test set for SGD and Adam optimizers: IID-40

4.7.4 Experiment V

As observed in section 4.7.3, attention model with bi-directional encoder and Adam optimizer with learning rate 0.001 performs the best so far. Till now, we have used greedy strategy with beam width 1 as decoding strategy and generated the translations. In this experiment, we use beam width of sizes: 5, 7, 9. We observe that beam width 5 shows an improvement over beam width of size 1. However, higher beam widths 7 and 9 do not show any improvement over beam width 5. Also, the translations obtained from beam widths 7 and 9 are the same as the ones obtained from beam width 5. One of the main takeaways of this section is that beam search with higher beam width doesn't necessarily improve the translations. So, continuously increasing the beam width may only increase the decoding time and not improve the results.

In Table 4.13, we show the translations which are different for the development set. The first translation improves drastically as it recognizes the base class correctly. For the second translation, it recognizes that there should be on more argument in the command even if the argument is incorrect. It becomes worse for the third translation as it doesn't even recognize the base class.

Table 4.13: Changed translations different beam widths in development set: IID-40

Input Description	**Translation Reference**	**Translation with beam width = 1**	**Translation with beam width = 5**
Take out the doc1.txt file from document.tat.bz2 bzip2 archive.	tar -xjf document.tar.bz2 "doc1.txt"	**sed** ⟨unk⟩ ⟨unk⟩	**tar** ⟨unk⟩ ⟨unk⟩
Overwrite file.txt with first 2 lines of the file lol.txt.	head -n 2 lol.txt ⟩ file.txt	head -q ⟨unk⟩ file.txt	head -q ⟨unk⟩ **user1** file.txt
Show only repeated lines from documents.txt.	uniq -d documents.txt	uniq -r test	**-n7** -r test

Table 4.14 shows a few translations which changed in the test set. The first translation improved as it recognized the correct base class "cat". It copies the correct argument

from the description adding to the argument accuracy. The last two translations show no improvement over greedy decoding strategy.

Table 4.14: Changed translations with different beam widths in test set: IID-40

Input Description	**Translation Reference**	**Translation with beam width = 1**	**Translation with beam width = 5**
Open file instructions.txt. Display non-printing characters as well as TAB character as ^I.	cat -t instructions.txt	wc -rf general.py	**cat -t**u general.py
Show last 500 bytes of /var/log/messages and lol.txt.	tail -c 500 /var/log/messages lol.txt	tail -c 500 500 rm	tail -c 500 500 **/var/log/messages**
Check if the file testsort.txt is sorted or not.	sort -c testsort.txt	sort -r dir1	sort -gp ⟨unk⟩
Invalidate timestamp of current user.	sudo -k	more ⟨unk⟩	free ⟨unk⟩

Table 4.13 and 4.14 show that the changes in translations are not guaranteed to be improvements.

4.8 Experiments and Results for COMPOSITIONAL-40

We perform the experiments listed in Table 4.1 for the COMPOSITIONAL-40 dataset and show the results. We show trends of training loss and development accuracy, along with n-gram and command distributions of test set.

4.8.1 Experiment I and II

As mentioned in Table 4.1, we experiment with two models: one containing uni-directional encoder and the other containing bi-directional encoder. We use stochastic gradient descent as optimizer with 4 learning rates: 0.01, 0.1, 0.5 and 1.0.

Performance on Development Set

Table 4.15 and 4.16 show best metrics obtained from both models. Table 4.15 shows that for a uni-directional, learning rate 0.5 contains the highest values for best validation accuracy and word accuracy. Values for best validation BLEU-7 score is 0.0 and best command accuracy is 0.7 achieved by 3 learning rates: 0.1, 0.5, 1.0. Table 4.16 shows that there is no one good SGD learning rate for bi-directional encoder. We choose to explore learning rate 0.5 as it has the highest validation accuracy and second highest validation word accuracy which is only 0.2 less than the best one.

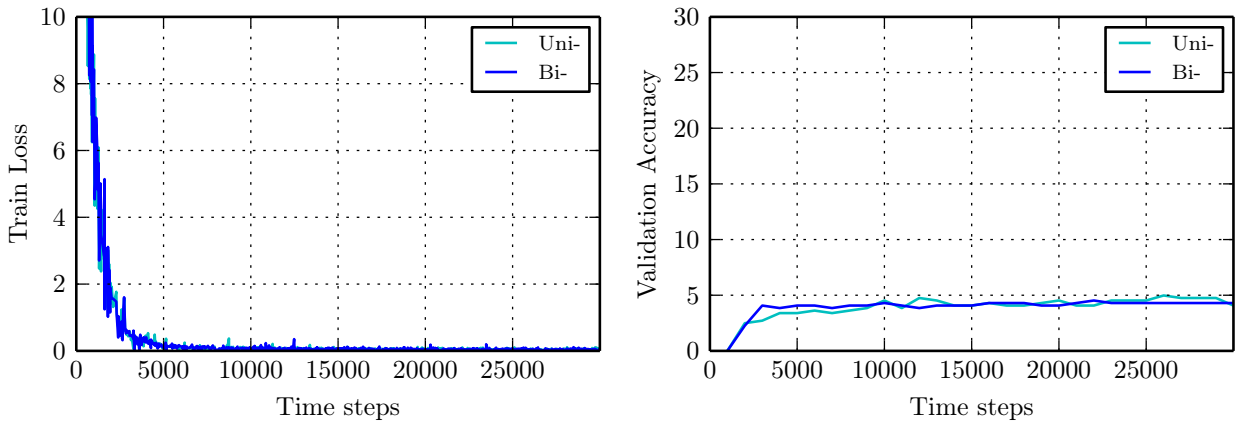
Table 4.15: Best Metric values for Uni-directional Encoder for COMPOSITIONAL-40

Learning Rate	**Best Development BLEU-7 Score**	**Best Validation Accuracy**	**Best Validation Word Accuracy**	**Best Validation Command Accuracy**
0.01	0.0	0.0	25.1	0.5
0.1	0.0	3.6	47.5	**0.7**
0.5	0.0	**5.0**	**48.4**	**0.7**
1.0	0.0	2.7	45.1	**0.7**

Table 4.16: Best Metric values for bi-directional encoder for COMPOSITIONAL-40

Learning Rate	Best Development BLEU-7 Score	Best Validation Accuracy	Best Validation Word Accuracy	Best Validation Command Accuracy
0.01	0.0	0.0	17.7	0.4
0.1	0.0	2.9	**47.6**	**0.7**
0.5	0.0	**4.5**	47.4	**0.7**
1.0	0.0	4.1	47.4	**0.7**

Figure 4.10 shows a performance comparison of training for models with uni- and bi-directional encoder with their optimal learning rates, 0.5. We observe that both models show the same trends for training loss and validation accuracy with almost equal performance values. A bi-directional model doesn't have a particular advantage over a uni-directional model.



(a) Training Loss

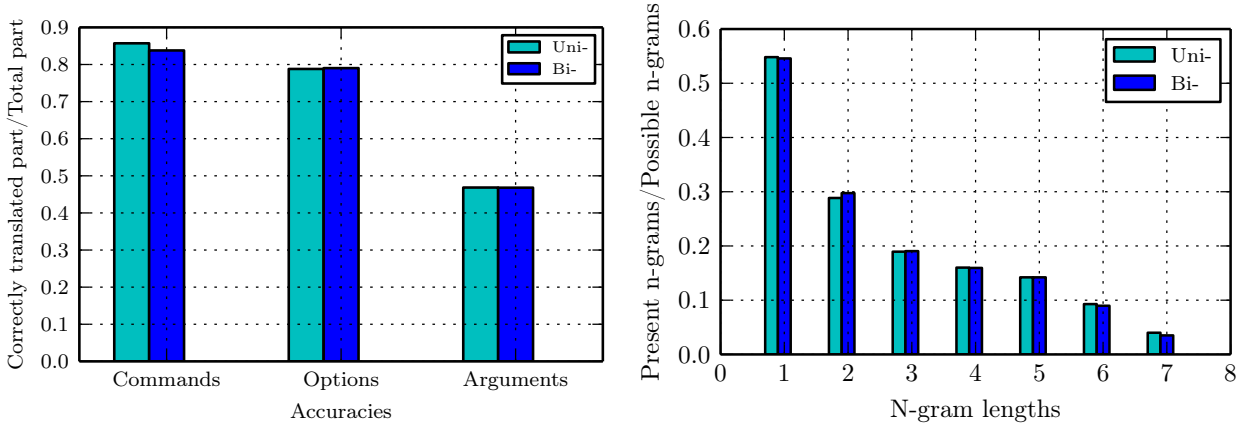
(b) Validation Accuracy

Figure 4.10: Training Parameters of uni- and bi- directional encoder models: COMPOSITIONAL-40

Performance on Test Set

Figure 4.11 confirms the same observations on the test set: both models perform equally well on this dataset. A uni-directional encoder performs 0.4% better on learning n-grams

of size 7 and 1.9% better on learning base classes. A bi-directional encoder performs 0.22% better in learning options parts of the syntax. Both encoders perform equally well on learning arguments.



(a) Command Distribution

(b) N-gram Distribution

Figure 4.11: Distributions of Test set with uni- and bi- directional encoders: COMPOSITIONAL-40

Results of Test set

A uni-directional model learns a total of 67 translations out of 882 verbatim, 38 of which are unique. A bi-directional encoder learns a total of 19 translations accurately, 10 of which are unique. A uni-directional encoder learns 36 commands of base class mount, 22 of class umount and the rest of chmod, bzip2 and cp. A bi-directional encoder learns 63 commands out of 882 verbatim, 37 of which are unique. Out of the unique commands, 39 are of base class mount, 11 are of class umount, rest are of bzip2, free, cd and tail. *Both models learn the same 27 commands accurately.* This shows that both models behave very similarly. Table 4.17 and 4.18 show the unique translations learned by both models. It should be noted that the maximum number of correct translations belong to classes have large number of data points. Hence, to get compositional generality, a lot of data for each class would provide good results.

Table 4.17: Verbatim translations with uni-directional encoder: COMPOSITIONAL-40

Input Description	Translation
Mount the file system /mydata in verbose mode. Tolerate sloppy mount options rather than failing. Don't call the /sbin/umount.⟨ filesystem ⟩ helper even if it exists.	mount -v -i -s /mydata
Give permission to all users of the group to read, write and execute the file filename. Generate a log. Prevent error messages from being shown.	chmod -vf 070 filename
Forcefully unmount the file system /mnt without writing in /etc/mtab. If /mnt is a loop device, also free this loop device.	umount -n -d -f /mnt

Table 4.18: Verbatim translations using bi-directional encoder: COMPOSITIONAL-40

Input Description	Translation
Mount the file system /mydata read/write without writing in /etc/mtab in verbose mode. Tolerate sloppy mount options rather than failing.	mount -v -n -s -w /mydata
Do a lazy unmount of the file systems of the type test in verbose mode.	umount -v -l -t test
Give user full access to filename while group can only read and execute and other users can read. Generate a log if changes are made to the file.	chmod -c 754 filename

Table 4.19 show different types of translations where either of the models perform same, better and worse.

4.8.2 Experiment II and III

As mentioned in Table 4.1, we experiment with two models: bi-directional encoder with and without attention. The attention model used is described in section 4.3. We use stochastic gradient descent as an optimizer with learning rate 1.0.

Table 4.19: Translations of both encoders: COMPOSITIONAL-40

Input Descriptions	**Reference Translation**	**Translation using uni-directional encoder**	**Translation using bi-directional encoder**
Ignore first 2 fields as well as the case when comparing lines and display only continuous repeated lines from the file sample.txt.	uniq -d -f 2 -i sample.txt	uniq -d -f 2 sample.txt	uniq -w -w 8 tutorial.txt
Mount file systems of the type test read-only in verbose mode. Tolerate sloppy mount options rather than failing.	mount -v -s -r -t test	mount -v -s -r -t test	mount -v -s -t test
Show help for less command.	less –help	getfacl -h	getfacl -h
Arrange the content of file blag.doc in reverse order and NULL is the line terminator.	sort -rz blag.doc	sort -bd -v now	sort -z blag.doc

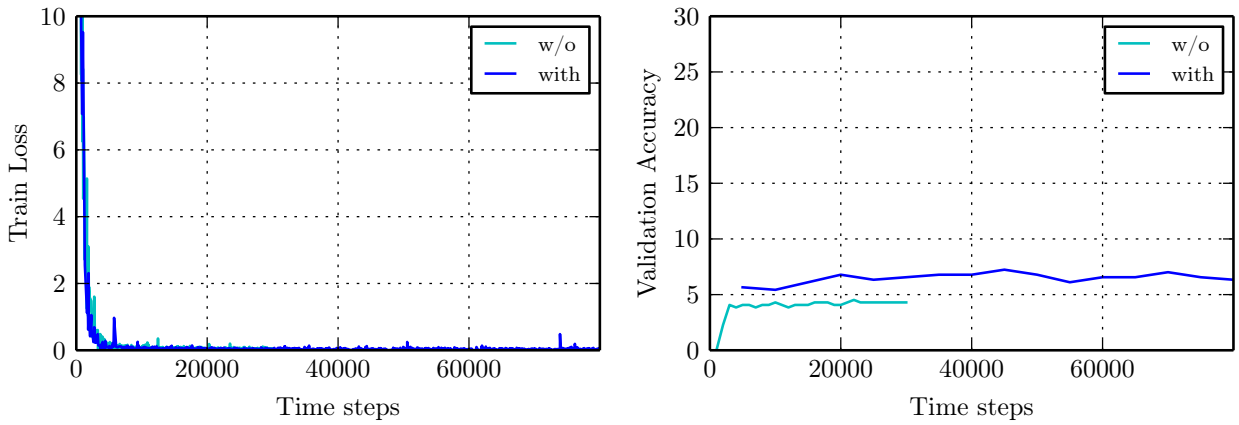
Performance on Development set

Table 4.20 shows a comparison of best metric values for model with attention using learning rate of 1.0 and model without attention with learning rate of 0.5. We have run the model with attention for 80,000 time steps and the model without attention for 30,000 time steps. We observe that the best metric values are obtained by the attention model.

Table 4.20: Best Metric values for model with and without attention: COMPOSITIONAL-40

Attention type	**Best Development BLEU-7 Score**	**Best Validation Accuracy**	**Best Validation Word Accuracy**	**Best Validation Command Accuracy**
without	0.0	4.5	47.4	0.7
with	0.0	**7.2**	**49.8**	0.7

Figure 4.12 shows that an attention model has the same trend for training loss as one without. However, attention model has a higher validation accuracy across training.



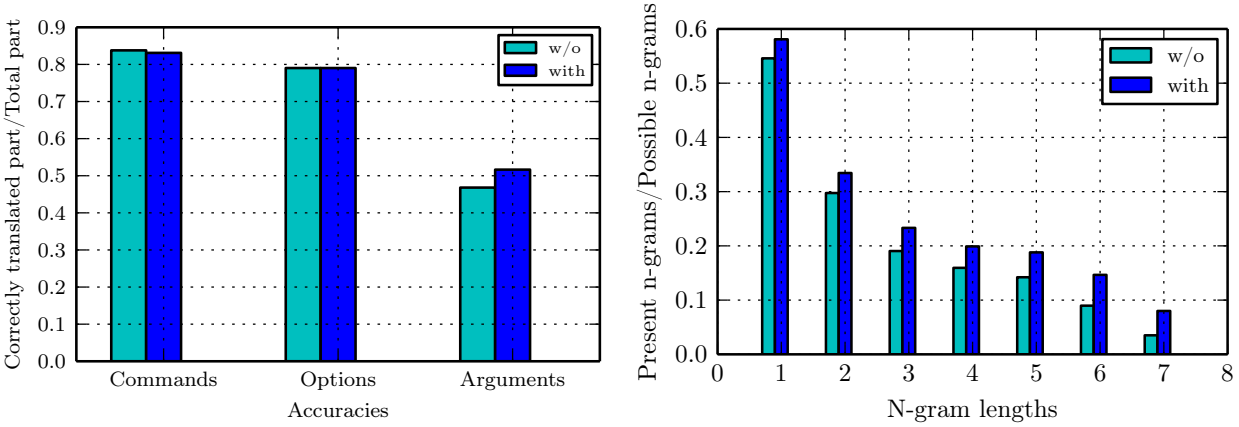
(a) Training Loss

(b) Validation Accuracy

Figure 4.12: Training Parameters of models with and without attention: COMPOSITIONAL-40

Performance on Test Set

Figure 4.13(a) shows that model with attention learns arguments of a command better than the model without attention by 4.83% percent. However, the model without attention learns base commands better by 0.68% percent. Figure 4.13(b) shows that the attention model learns n-grams of all sizes(1-7) better than model without attention, with n-grams of size 7 better by 4.49%.



(a) Command Distribution

(b) N-gram Distribution

Figure 4.13: Distributions of Test set using models with and without attention: COMPOSITIONAL-40

Results of Test set

The attention model learns 90 commands accurately out of 882 in the test set out of which 11 are unique. Among all accurate translations, 44 are of mount, 27 are of umount, 11 are of bzip2 and the rest belong to chmod, cd, tail, find etc. Table 4.21 shows the accurate translations (exact copies of their reference) generated by attention model on the test set.

Table 4.21: Verbatim translations with attention model: COMPOSITIONAL-40

Input Descriptions	**Translation**
Give permission to all non group users to read, write and execute the file filename. Generate a log. Prevent error messages from being shown.	chmod -vf 007 filename
Search for all the files whose permission is 0745 and has access time newer than the modification time of tilda.txt file.	find . -perm 0745 -anewer tilda.txt
Do a lazy unmount of the file system /myfol without writing in /etc/mtab. If the unmount fails, try to remount read-only.	umount -n -r -l /myfol

Table 4.22 shows translations of both models and their performance on different descriptions. In the first translation, the attention model translates the arguments correctly and the other model cannot. However, in the fifth translation, the model without attention copies arguments correctly. The second and third translations show the same reference command being translated by two different descriptions by both models. For one description, the model without attention translates accurately and for another description, the model with attention does. Two examples are shown where neither model translates properly.

Table 4.22: Translations using models with and without attention: COMPOSITIONAL-40

Input Description	**Reference Translation**	**Translation with attention**	**Translation without attention**
Show the differences between one.txt and two.txt files in context format. Do not distinguish lines on the basis of number of white space characters.	diff -cb one.txt two.txt	diff ⟨unk⟩ one.txt two.txt	diff ⟨unk⟩ abc.txt xyz.txt
Change current environment to /full/path/to/f*.	cd /full/path/to/f*	cd /full/path/to/f*	cd /relative/path/to/folder
Command to change directory to /full/path/to/f*.	cd /full/path/to/f*	cd /relative/path/to/folder	cd /full/path/to/f*
Name all contents, except the backup files, of the directory Tones, line by line.	ls -B -x Tones	find . -type l -name ⟨unk⟩	ls -x ⟨unk⟩
Create symbolic link file2.txt to point to file1.txt. If an existing destination file cannot be opened, remove it and try again.	cp -f -s file.txt file2.txt	cp -f -s /myfiles/file*.txt /myfiles2	cp -i -s -v file.txt file2.txt
Locate all files with modification timestamp of nearly 5 days ago and the user owner is soul in /manga directory.	find /manga/ -user soul -mtime 5	find /manga/ -user soul -atime 5	find ⟨unk⟩ -maxdepth l /etc/

4.8.3 Experiment III and IV

As mentioned in Table 4.1, we experiment with Adam optimizer using different learning rates: 0.01 and 0.001 on a bi-directional model with attention.

Performance on Development set

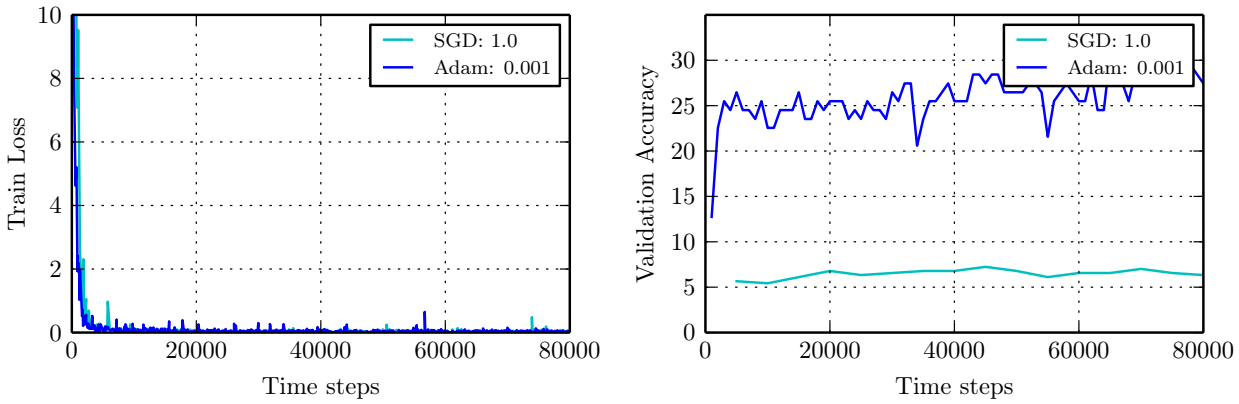
Table 4.23 shows the performance of two learning rates 0.01 and 0.001 of Adam optimizer in an attentional model with bi-directional encoder. We can observe that learning rate 0.001 performs way better than 0.01 for all metrics, except command accuracy. We compare the

training of models Adam optimizer with learning rate 0.001 and stochastic gradient descent with learning rate 1.0.

Table 4.23: Best Metric values using Adam optimizer for COMPOSITOIONAL-40

Learning Rate	Best Development BLEU–7 Score	Best Validation Accuracy	Best Validation Word Accuracy	Best Validation Command Accuracy
0.01	15.4	11.1	55.8	**0.8**
0.001	**43.3**	**30.4**	**71.2**	**0.8**

Figure 4.14 show how both models perform during training phase. We see that both models show the same trend for decrease in train loss. For validation accuracy, we see a stark difference with Adam optimizer showing 20% higher accuracy than SGD.



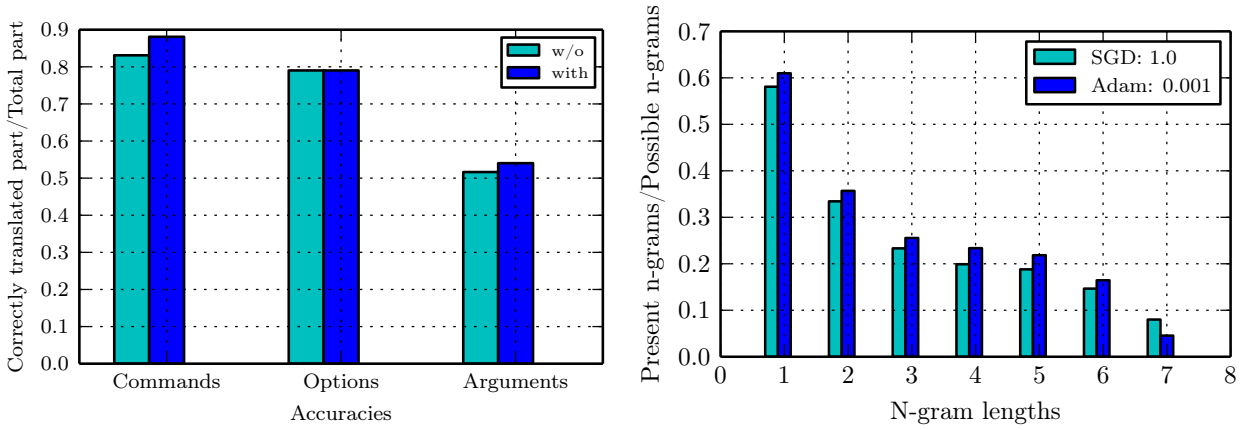
(a) Training Loss

(b) Validation Accuracy

Figure 4.14: Training Parameters for different optimizers: COMPOSITIONAL-40

Performance on Test set

Figure 4.15 confirms our observations that Adam optimizer performs better than SGD. Adam optimizer performs 4.98% better on translating base Linux commands and 2.39% on copying arguments into the translation. Both models learn n-grams of size 7. It's interesting to note that model with SGD learn n-grams of size 7 with 3.45% better than ADAM optimizer.



(a) Command Distribution

(b) N-gram Distribution

Figure 4.15: Distributions of Test set for SGD and Adam optimizers: COMPOSITIONAL-40

4.8.4 Experiment V

As observed in section 4.8.3, attention model with bi-directional encoder and Adam optimizer with learning rate 0.001 performs the best so far. Till now, we have used greedy strategy with beam width 0 as decoding strategy and generated the translations. In this experiment, we use beam width of sizes: 5, 7, 9. We observe that beam width 5 shows an improvement over greedy strategy. However, higher beam widths 7 and 9 do not show any improvement over beam width 5. Also, the translations obtained from beam widths 7 and 9 are the same as the ones obtained from beam width 5. Same observation is made for IID-40 dataset. A higher beam width doesn't necessarily improve the quality of translations.

Table 4.24 shows the translation which changed in the validation set from beam width 1 to beam width 5. We can see that the translation is incorrect after decoding with beam width 1 and doesn't improve when higher beam width of 5 is used.

Table 4.25 shows the translations which change in the test set when a higher beam width of 5 is used for decoding. The first translation gets only the base class correct. It doesn't improve with beam width 5. The second translation doesn't translate all the options correctly and doesn't improve with a higher beam width. The third translation doesn't get

even the base class correct and doesn't improve at all.

Table 4.24: Translations with different beam widths in validation set: COMPOSITIONAL-40

Input Description	**Translation Reference**	**Translation with beam width = 1**	**Translation with beam width = 5**
Make the owner, group and other users have no permissions to the file filename, prevent error messages from being shown and generate a log if changes are made to the file.	chmod -cf 000 filename	less +5 $\langle unk \rangle$ hello list.txt	less $\langle unk \rangle$ $\langle unk \rangle$ list.txt

Table 4.25: Translations with different beam widths in test set: COMPOSITIONAL-40

Input Description	**Translation Reference**	**Translation with beam width = 1**	**Translation with beam width = 5**
Check the total file size of the files in the folder containing the file extension .out.	du -s *.out	du $\langle unk \rangle$ -cwzf anime.tar.gz $\langle unk \rangle$	du /home/user/ $\langle unk \rangle$ root sortedbig.txt
Backup existing file named dst˙link.txt and remove the same. Make link from dst˙link.txt to the symlink src˙original.txt in current directory. It should be a hard link, not soft. Also, give a printed confirmation for the same on the command line.	ln –backup -P -f -v src˙original.txt dst˙link.txt	ln $\langle unk \rangle$ $\langle unk \rangle$ -v src˙original.txt dst˙link.txt	ln $\langle unk \rangle$ -i -v src˙original.txt dst˙link.txt
Show distinctions between WINDOWS and BSD recursively, and turn off case sensitivity for file names.	diff –ignore-file-name-case -r WINDOWS BSD	export $\langle unk \rangle$ $\langle unk \rangle$ $\langle unk \rangle$	export -x $\langle unk \rangle$ $\langle unk \rangle$ $\langle unk \rangle$

CHAPTER 5
CONCLUSION

This thesis presented a parallel corpus containing executable Linux commands and natural language descriptions describing their function comprehensively and established baseline approaches for classifying the intended command and producing the intended command using sequence to sequence models. We obtained promising results which indicate that with only natural language as input, the main Linux command to be used can be learnt with high accuracy, overall syntax of different Linux commands and executable commands producing correct results can be learnt. In future work, it would be interesting to perform translations grounded in the Linux execution environment where one trains the translation to be aware of how it changes the state of the Linux system and to lead to the correct/intended set of changes.

Appendices

APPENDIX A

LINUX COMMANDS

The set of Linux commands given to each user is shown in Table A.1.

Table A.1: Users and Commands

User ID	Commands Created
1	cp, free, ifconfig, mount, umount, ln, ls, uniq
2	service, shutdown, chmod, du, df, wget
3	head, tail, su, whatis, whereis, passwd
4	diff, find, mv, sort, tar, ssh
5	bzip2, chown, sed, export
6	less, getfacl, sleep
7	rm, mkdir, cat, awk, more, sudo, wc, unzip, ps, cd
8	echo, history, kill, pwd

Following is a list of all commands shown in Table A.1, along with their usage, descriptions and an example:

1. **AWK**

   Using this command, we can select particular records in a file and perform operations upon them. Out of all options supported by this command, we used: -F, -f. A sample:

   Command: awk 'print' marks.txt

   Description: Show the contents of marks.txt.

2. **BZIP2**

   bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. The options used along with this command: -c,-d,-z,-f,-k,-s,-v,-q,–help,-V, -1,-2,-3,-4,-5,-6,-7,-8,-9,-t. An example pair:

   Command: bzip2 -4 -k file.txt

Description: Compress the file file.txt with level 4 compression and don't delete file.txt.

3. **CAT**

   This command concatenates files and prints the result on the standard output. Options used for creating commands: -n,-b,-e,-E,-s,-v,-t,-T,-u. Following is an example:

   Command:cat -bsvu instructions.txt

   Description: Open file instructions.txt. Display line numbers for non empty lines. Print non-printing characters. Squeeze adjacent multiple empty lines. Set output buffering disabled.

4. **CD**

   The cd command is used to change the current directory (i.e., the directory in which the user is currently working) in Linux and other Unix-like operating systems. Only two options are supported by this command: -P,-L. Example:

   Command: cd -L /relative/path/to/folder

   Description: Make my current workspace to /relative/path/to/folder. Force symbolic links to be followed.

5. **CHMOD**

   This command changes access permissions/mode for files and folders. Options supported by this command: -c,-f,-v,-r, –no-preserve-root,–preserve-root,–reference-file. Mode of permissions can be expressed numerically or with a character.

   For example:

   Command: chmod -Rv 755 directory-name/

   Description: Give permission to all the content of the folder directory-name such that the user can have full access while group and other users can read and execute. Generate a log.

Table A.2: Default Permissions

No.	Permissions	Characters
7	read,write,execute	rwx
6	read,write	rw-
5	read,execute	r-x
4	read only	r–
3	write, execute	-wx
2	write only	-w-
1	execute	–x

6. **CHOWN**

   The chown command is used to change the ”owner” and ”group” of files, directories and links. It uses the default permission symbols of UNIX Commands. Options used for creating commands: –from, -c,-f,-v,-h,–dereference, –reference,-R, -H, –version, –help. An example command:

   Command: chown -f –from=user1 :group1 file.txt

   Description: If the current user is known to be user1, change the group ownership of file.txt to group1. Don’t display any error/warnings.

7. **CP**

   The cp command is used to make copies of files and directories. the options using which commands were created are -a, –attributes-only, –backup, -b, -f, -i, -L, -l, -n, -P -r, –remove-destination, -s, -u, -v. Example of a command and description:

   Command: cp -b *.txt ../office

   Description: Create a backup of each existing destination file in office folder in parent directory while copying all files ending with .txt.

8. **DF**

   The df command stand for disk filesystem. It is used to get full summary of available and used disk space usage of file system on Linux system. the options used are -a, -b, -h, -h, -i, -k, -l, -P, –total, -T, -x, -v. Example:

   Command: df -BK -a

   Description: Show statistics of all file systems in terms of Kilobytes along with dummy ones.

9. **DIFF**

   This command compares the contents of two files and write to the standard output the list of changes necessary to convert one file into the other. Out of the options supported by this command, the following is used: -u, -i, -q, -s, -c, -e, -y, –width, -b, -Z, -w, -b, -n, -p, –speed-large-files, –strip-trailing-cr, -D, –color, -E, -N, -l, -I, –suppress-common-lines, –left-column, –ignore-file-name-case, –from-file,–to-file,-r, -s, -x, -X, –tabsize, -C, -u, –palette, –no-dereference, -t, –version, –help. For example:

   Command: diff -ubBdl abc.txt xyz.txt

   Description: Show the minimal differences between abc.txt and xyz.txt ignoring blank line changes and amount of white space in unified format. Paginate the output.

10. **DU**

    DU is a standard Unix program used to estimate file space usagespace used under a particular directory or files on a file system. The options used to create complete commands for du: -a, -b, -B, -c, -d, -h, –inodes, -k, -m, –si, -t, –time, –time-style, -x, -s. Example:

    Command: du -s -m /home/user/

    Description: Displays the total file size of the files in in /home/user/ in terms of

Megabytes.

11. **ECHO**

    Echo is a built-in command in the bash and C shells that writes its arguments to standard output. Echo has only a few options: -e, -E and -n which are used. Following is an example.

    Command: echo -n 'Hello World'

    Description: Display string Ḧello Worldón command line without a newline inserted at the end of the output.

12. **EXPORT**

    Export is a built-in command of the bash shell and other Bourne shell variants. It is used to mark a shell variable for export to child processes. Export doesn't have many options. Example for export command:

    Command: export VAR=1

    Description: What is the command to make a variable VAR and equate itś value to 1?

13. **FIND**

    In Unix-like and some other operating systems, find is a command-line utility that searches one or more directory trees of a file system, locates files based on some user-specified criteria and applies a user-specified action on each matched file. Options of find used to create complete commands are: -maxdepth, -not, -name, -type, -perm, -user, -group, -mtime, -atime, -cmin, -ctime, -mmin, -cmin, -size, -empty, -iname, -newer, -mindepth, -inum, -anewer, -cnewer, -xdev, -amin, -anewer, -inum, –version. An example for find command:

    Command: find / -maxdepth 2 -type d -name '*.pl'

    Description: List all those directories which end with .pl in / directory. Search only

till the directory depth of 2.

14. **FREE**

    Free displays the total amount of free and used physical memory and swap space in the system, as well as the buffers and cache consumed by the kernel. The options using which commands are created: -m, -b, -k, -g, –tera, -h, -si, -l, -o, -t, -s, -c.

    Command: free -l -b –si

    Description: How much free memory do I have on my disk in bytes? Show detailed low and high memory statistics. Use powers of 1000 not 1024.

15. **GETFACL**

    For each file, getfacl displays the file name, owner, the group, and the Access Control List (ACL). If a directory has a default ACL, getfacl also displays the default ACL. The options used: -a, -n, -d, -c, -e, -E, -s, -R, -t, -v, -h.

    Command: getfacl -ER ../sample/files

    Description: Show all information about all the files and folders in the directory ../sample/files without the comments for effective rights.

16. **HEAD**

    The head command reads the first few lines of any text given to it as an input and writes them to standard output (which, by default, is the display screen). The options used: -n, -c, -q, -v, –help, –version.

    Command: head -q -c 80k instructions.txt

    Description: Show the starting 80 x 1024 characters of the file instructions.txt without header.

17. **HISTORY**

    The various Unix shells maintain a record of the commands issued by the user during the current session. The history command manipulates this history list. In its simplest

form, it prints the history list. Options allow for the recall and editing of particular commands and for setting parameters such as the number of past commands to retain in the list. Options are -a, -c. An example is provided:

Command: history 5

Description: Show the last 5 commands executed in this session.

18. **IFCONFIG**

Ifconfig is used to configure the kernel-resident network interfaces. It is used at boot time to set up interfaces as necessary. After that, it is usually only needed when debugging or when system tuning is needed. The options of ifconfig command used are: -a -s, -v, -promisc, -arp, -allmulti, -broadcast, -pointopoint. For example:

Command: ifconfig -v eth0

Description: How to see the verbose details of error conditions of eth0?

19. **KILL**

This command sends a signal to one or more processes (usually to terminate a process). It has only three options, only one of which we have used: -l. The arguments for this command contain the signal to send to a particular process. For example,

Command: kill 485

Description: How do I end process having PID 485?

20. **LESS**

less is a terminal pager program on Unix, Windows, and Unix-like systems used to view (but not change) the contents of a text file one screen at a time. It is similar to more, but has the extended capability of allowing both forward and backward navigation through the file. It contains many options of which we have created commands using the following: -a, -b, -E, -f, -F, -g, -G, -k, -i, -I, -j, -J, -m, -M, -n, -N, -r, -s, -S, -u, -p. For example:

Command: less -Ip hello$ list.txt

Description: Open list.txt in command line, one pageful at a time. While searching, highlight the locations where the hello word was at the end of string or line. Ignore case while searching.

21. **LN**

The ln command is a standard Unix command utility used to create a hard link or a symbolic link to an existing file. Combinations of options -s, –backup, -d, -f, -i, -P, -s, -v, -t, –help, –version were used to create commands. For example:

Command: ln –backup -s -i -v ../Batch_with_3 docs

Description: Backup current file named docs. Create a soft link named docs in the current directory that points to the contents of the folder Batch_with_3 present in the parent directory. Ask permission before removing docs if it already exists in the folder. Print the name of each linked file.

22. **LS**

This command lists all files in a UNIX/UNIX-like operating system. The options of ls command used are: -R, -l, -t, -a, -h, -F, -S, -i, -n, -1, -m, -s, -r, -B, -f, -x, -U. Example:

Command: ls -lt -c Juices

Description: Show a detailed list of files and folders in the Juices directory and sort them by time of last modification of file status, newest first.

23. **MKDIR**

This command creates a new folder using command line, if it doesn't exist. The options used to create commands for mkdir are -m, -v, -p, -Z.

Command: mkdir -vp myfolder/main/ananya

Description: Create folder ananya and it's parent directories(myfolder/main) with a

log.

24. **MORE**

    The more command allows you to display output in the terminal one page at a time. -v, –help, -p, -u, -d, -f, -s, -c, -l are the options used.

    Command: more -cd -10 myfile.txt

    Description: How to clear the current screen and display contents of myfile.txt with prompt instead of ringing bell when an illegal key is pressed and screen size set as '10'?

25. **MOUNT**

    The mount command mounts a storage device or filesystem, making it accessible and attaching it to an existing directory structure. -t, -l, -M, -h, -v, -V, -a, –fork, –fake, -i, -n, -p, -s, -r, -w, -L, -U, –no-canonicalize options are used to create commands. Example:

    Command: mount -v -i -l -s /mydata

    Description: Single command to mount the file system /mydata in verbose mode. Tolerate sloppy mount options rather than failing. Add the labels in the mount output. Don't call the /sbin/umount.⟨ filesystem ⟩ helper even if it exists.

26. **MV**

    The mv command moves, or renames, files and directories on your filesystem. The options used to create the commands are –backup, -f, -i, -n, -b, -S, -v, -u, -Z, -t, –strip-trailing-slashes, -T. Example:

    Command: mv -nvZ bleach.txt anime/

    Description: Move bleach.txt -⟩ anime/bleach.txt and do not over-write. Set SEL security context to default and show the jobs completed.

27. **PASSWD**

The passwd command is used to change the password of a user account. A normal user can run passwd to change their own password, and a system administrator can use passwd to change another user's password, or define how that account's password can be used or changed. Options used: -S, -a, -u, -l, -e, -d, -n, -x, -w, -q, -d, -n. Example:

Command: sudo passwd -e jane

Description: Expire password of username jane.

28. **PS**

The ps, abbreviation for process status, is a command that provides information about the currently running processes. It supports many options out of which -e, -f, -U, -g, -p, –ppid, -U, -L, -C, -G are used. For example,

Command: ps -f -U 1234

Description: How do I see the state of all processes run by user id 1234?

29. **PWD**

The pwd command reports the full path to the current directory. This command supports only 2 options -L and -P, which are used in creating commands. For example,

Command: pwd -P

Description: Print current physical directory without considering symbolic links.

30. **RM**

The command rm is used to remove objects such as files, directories, device nodes, symbolic links etc, from the file system. Options used to create commands: -f, -I,-d, -r, -v. For example,

Command: rm -vi linuxfile.log

Description: Remove file linuxfile.log. Show what is being removed and confirm before removing.

31. **SED**

Command sed, short for stream editor, is used to perform operations on a file such as insert, delete, substitute, find etc. Complete commands are created using -n, -i, -e, -r. Example:

Command: sed 's/tom/larry/gi' new.txt *Description*: How do I replace tom with larry in new.txt without worrying about case sensitivity?

32. **SERVICE**

Service command can be used to start, stop, and restart the daemons and other services under Linux. It contains only one option, –status-all which has been used. For example,

Command: service stop httpd

Description: How can I make the system to stop the httpd service?

33. **SHUTDOWN**

The shutdown command brings the system down in a secure way. It supports only 4 options namely, -h, -c, -r, -t. Example of a command created with shutdown:

Command:shutdown -t 10

Description:Shuts down the system after a 10 second delay.

34. **SLEEP**

The sleep instruction suspends the calling process for at least the specified number of seconds, minutes, hours or days. It contains no options, just suffixes to represent the time frame of suspension required. For example:

Command: sleep 18m 9d 15d

Description: Make command line pause for 18 minutes , 9 days and 15 days.

35. **SORT** Sort command rearranges the lines in a text file so that they are sorted, numerically and alphabetically. The commands are created using many options namely: -f, -k, -r, -n, -u, -h, -i, -M, -z, -m, -c, -t, -d, -V, –files0-from,-T, –compress-program, -b, -B, –random-source, –parallel, -R, -S. Example:

    Command:sort -ur dupl.txt

    Description:How do I get the content of file dupl.txt reverse sorted and only unique lines to be shown?

36. **SSH** The ssh command provides a secure encrypted connection between two hosts over an insecure network. This connection can also be used for terminal access, file transfers, and for tunneling other applications.

    Command:ssh -1 pratik.jain@web.iiit.ac.in

    Description:How do I login to remote server web.iiit.ac.in using username pratik.jain using protocol v1 ?

37. **SU** The Unix command su, sometimes described as substitute user, super user, or switch user, is used by a computer user to execute commands with the privileges of another user account. the options used to create the commands are: -c, –shell, -m, -l. Example:

    Command:su -m guest -c date

    Description:Login as username guest while preserving my current account, show date and time.

38. **SUDO**

    The command sudo, ”superuser do”, allows a user with proper permissions to execute a command as another user, such as the superuser. The options used for creating the commands are: -b, -V, -h, -K, -v, -g, -u. For example:

    Command: sudo -K

Description:Surely kill timestamp of current user.

39. **TAIL**

    The tail command is a command-line utility for outputting the last part of files given to it via standard input. The options used to create tail commands are: -c, -f, -n, -v, -s, -q.

    Command: tail -n7 lol.txt

    Description: Command to see last 7 lines of lol.txt.

40. **TAR**

    It is an abbreviation of "Tape Archive". It is used to create, extract and list files from tar files. The options used while creating commands for tar are: -c, -v, -z, -f, -x, -j, -t, -w, -r, -W, –delete, -A, –test-label, -u, -d, –no-recursion, –xattrs, –no-attrs, –selinux, –no-selinx, –exclude, –wildcards.

    Command:tar -cvf test.tar test_folder

    Description: Make of the test_folder with name test.tar displaying the files added.

41. **UMOUNT** The umount command detaches the file system(s) mentioned from the file hierarchy. The options used to create executable umount commands are: -v, -n, -i, -a, -r, -d, -t, -f, -l, -h, -V.

    Command: umount -r /mydata

    Description:Single command to unmount the file system /mydata and remount read-only if the unmount fails.

42. **UNIQ** The uniq command reports/filters out repeated lines in a file. The options used to create executable commands for uniq are:

    Command:uniq -c -w 8 -i tutorial.txt

    Description:Don't consider the first 8 characters and the case while comparing lines

and display only unique lines from the file tutorial.txt on the command line along with it's number of occurences.

43. **UNZIP** The Linux command unzip will list, test, or extract files from a ZIP archive. The options -d, -x, -p, -f, -u, -v, -l, -t, -z, -T, -n, -o, -j, -U, -C, -K, -X, -q, -a, -L are used to create complete commands.

    Command: unzip -oq -UU letters.zip

    Description: Uncompress letters.zip quietly , ignore any unicode named file names and overwrite existing files.

44. **WC**

    The wc command, short for Word Count, is used to find out number of newline count, word count, byte and characters count in a file specified by the file arguments on a Linux/Unix system. The options used for creating the commands re -l, -w, -m -c, -l, -L.

    Command: wc -wc general.py

    Description:Show total number of words and total number of bytes in the file general.py.

45. **WGET** The wget is a computer program that retrieves content from web servers. It is part of the GNU Project. Its name derives from World Wide Web and get. It supports downloading via HTTP, HTTPS, and FTP. The options used to create commands are: -q, –input-file, –base, -c, -r, –output, –report-speed, –tries, –wait.

    Command:wget pes.edu/logo.png –tries=3

    Description: How do I try to download the logo.png resource from the pes.edu website 3 times?

46. **WHATIS**

The whatis command is helpful to get brief information about Linux commands or functions. Whatis command displays man page single line description for command that matches string passed as a command line argument to whatis command. The options -s, -r, -V, -l are used to create the executable commands for whatis.

Command:whatis -r ls

Description:Print all commands with 1 line descriptions which has ls in their names.

47. **WHEREIS**

Whereis command is helpful to locate binary, source and manual pages of commands in the Linux system. The options which are used to create commands are: -b, -m, -s, -B, -V, -h, -b,-M, -S.

Command:whereis -b -m python

Description:Locate all locations of binary files and manual files of python.

REFERENCES

[1] R. J. Waldinger and R. C. T. Lee, "Prow: A step toward automatic program writing," in *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, ser. IJCAI'69, Washington, DC: Morgan Kaufmann Publishers Inc., 1969, pp. 241–252.

[2] Z. Manna and R. Waldinger, "A deductive approach to program synthesis," *ACM Trans. Program. Lang. Syst.*, vol. 2, no. 1, pp. 90–121, Jan. 1980.

[3] J. K. Feser, S. Chaudhuri, and I. Dillig, "Synthesizing data structure transformations from input-output examples," *SIGPLAN Not.*, vol. 50, no. 6, pp. 229–239, Jun. 2015.

[4] S. E. Reed and N. de Freitas, "Neural programmer-interpreters," *CoRR*, vol. abs/1511.06279, 2015. arXiv: 1511.06279.

[5] C. Li, D. Tarlow, A. L. Gaunt, M. Brockschmidt, and N. Kushman, "Neural program lattices," *ICLR*, 2017.

[6] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *CoRR*, vol. abs/1410.5401, 2014. arXiv: 1410.5401.

[7] P. Bielik, V. Raychev, and M. Vechev, "Phog: Probabilistic model for code," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., ser. Proceedings of Machine Learning Research, vol. 48, New York, New York, USA: PMLR, 2016, pp. 2933–2942.

[8] M. Allamanis and C. A. Sutton, "Mining source code repositories at massive scale using language modeling," in *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*, 2013, pp. 207–216.

[9] V. Raychev, M. T. Vechev, and A. Krause, "Predicting program properties from "big code"," in *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, 2015, pp. 111–124.

[10] A. T. Nguyen and T. N. Nguyen, "Graph-based statistical language model for code," in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, 2015, pp. 858–868.

[11] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep API learning," *CoRR*, vol. abs/1605.08535, 2016. arXiv: 1605.08535.

[12] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Summarizing source code using a neural attention model," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[13] M. Allamanis, H. Peng, and C. A. Sutton, "A convolutional attention network for extreme summarization of source code," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, 2016, pp. 2091–2100.

[14] X. V. Lin, C. Wang, D. Pang, K. Vu, L. Zettlemoyer, and M. D. Ernst, "Program synthesis from natural language using recurrent neural networks," University of Washington Department of Computer Science and Engineering, Seattle, WA, USA, Tech. Rep. UW-CSE-17-03-01, Mar. 2017.

[15] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks.," *CoRR*, vol. abs/1409.3215, 2014.

[16] A. Neelakantan, Q. V. Le, M. Abadi, A. McCallum, and D. Amodei, "Learning a natural language interface with neural programmer," *CoRR*, vol. abs/1611.08945, 2016. arXiv: 1611.08945.

[17] W. Ling, E. Grefenstette, K. M. Hermann, T. Kociský, A. Senior, F. Wang, and P. Blunsom, "Latent predictor networks for code generation," *CoRR*, vol. abs/1603.06744, 2016. arXiv: 1603.06744.

[18] P. Yin and G. Neubig, "A syntactic neural model for general-purpose code generation," *CoRR*, vol. abs/1704.01696, 2017. arXiv: 1704.01696.

[19] K. Guu, P. Pasupat, E. Z. Liu, and P. Liang, "From language to programs: Bridging reinforcement learning and maximum marginal likelihood," *CoRR*, vol. abs/1704.07926, 2017. arXiv: 1704.07926.

[20] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[21] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[22] T. Kacmajor. (). Svm figure, url = http://tomaszkacmajor.pl/index.php/2016/04/24/svm-model-selection/, urldate = 2010-09-30.

[23] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *Trans. Neur. Netw.*, vol. 13, no. 2, pp. 415–425, Mar. 2002.

[24] S. Bird, E. Klein, and E. Loper, *Natural language processing with python*, 1st. O'Reilly Media, Inc., 2009, ISBN: 0596516495, 9780596516499.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[26] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014. arXiv: 1409.0473.

[27] P. Koehn, "Neural machine translation," *CoRR*, vol. abs/1709.07809, 2017. arXiv: 1709.07809.

[28] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016. arXiv: 1609.04747.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. arXiv: 1412.6980.

[30] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL '02, Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 311–318.

[31] M. Luong, E. Brevdo, and R. Zhao, "Neural machine translation (seq2seq) tutorial," *Https://github.com/tensorflow/nmt*, 2017.